

Masarykova univerzita v Brně  
Fakulta informatiky

Diplomová práce

## **JavaScript v příkladech**

Lenka Švancarová

Brno 2000



11.12.2000

Hilma Horáčková

Poděkování:

Děkuji **RNDr. Tomáši Pitnerovi Dr.** za ochotu, trpělivost a laskavou péči při vedení této diplomové práce. Dále děkuji **Mgr. Liboru Bednářovi** za pomoc a cenné rady, všem svým přátelům za psychickou podporu a rodičům za porozumění.

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pouze s využitím uvedené literatury.



V Brně, dne 9. prosince 2000

Lenka Švancarová

# **Obsah**

<b>OBSAH .....</b>	<b>3</b>
<b>ÚVOD.....</b>	<b>4</b>
<b>NÁVRH METODIKY .....</b>	<b>5</b>
<b>ÚVOD KE KURZU .....</b>	<b>6</b>
<b>KURZ JAVASCRIPTU.....</b>	<b>8</b>
PRVNÍ KROKY .....	9
SYNTAXE JAVASCRIPTU .....	14
UDÁLOSTI A JAVASCRIPT.....	28
OBJEKTY .....	32
OBJEKTY JAVASCRIPTU.....	43
CO SE NEVLEZLO JINAM.....	60
<b>ZDROJE JAVASCRIPTU.....</b>	<b>62</b>
<b>REJSTŘÍK.....</b>	<b>65</b>
<b>PŘÍLOHA A .....</b>	<b>66</b>
REZERVOVANÁ SLOVA JAVASCRIPTU.....	66
<b>PŘÍLOHA B .....</b>	<b>67</b>
UKÁZKA POUŽITÍ JAVASCRIPTU V PRAXI .....	67
<b>PŘÍLOHA C .....</b>	<b>70</b>
SROVNÁNÍ JAVASCRIPTU A PASCALU.....	70
<b>POUŽITÁ LITERATURA A ODKAZY.....</b>	<b>72</b>

# Úvod

Cílem této diplomové práce je navrhnout metodiku základního kurzu JavaScriptu a na základě ní kurz sestavit. Kurz JavaScriptu musí být realizován ve formě ucelené učebnice použitelné i jako distanční podpora. Od žáka tohoto kurzu se očekává předchozí základní znalost algoritmizace a programování a tvorby HTML stránek.

V první části navrhnu metodiku kurzu.

Další kapitola bude již přímo součást zmiňované učebnice a přinese její úvod, který se snaží motivovat čtenáře ke studiu JavaScriptu. Popisuje také veškeré vstupní požadavky na potencionálního studenta kurzu.

Ve třetí části najdete samotný kurz JavaScriptu. Snažila jsem se v něm respektovat mnou navrženou metodiku. Krom nezbytné teorie obsahuje ilustrační příklady a zejména cvičení.

Samostatnou kapitolou je závěr knihy, jež je věnován zejména těm, které JavaScript zaujal natolik, že mají zájem se o něm dozvědět něco více. Mimo jiného zde najdete odkazy na informační zdroje s tímto tématem.

Přílohy A, B a C jsou rovněž součástí učebnice JavaScriptu. V první jsou rezervovaná slova, druhá pojednává o složitějším skriptu (najdete jej na přiloženém médiu) a třetí se snaží o srovnání JavaScriptu s Pascalem. Poslední příloha je zařazena zejména s ohledem na to, že velká část samostudujících se setkala patrně někdy v minulosti s Pascalem.

## Návrh metodiky

Pokud má učební text sloužit zejména k samostudiu, pak je nutné v jeho úvodu uvést, co by měl čitatel znát, aby jej byl schopen pochopit. Samotná metodika by pak měla být zvolena tak, aby student nebyl pouze pasivním čtenářem. Nejvhodněji se mi jeví varianta, kdy učební text obsahuje kromě teoretického základu příklady a cvičení.

Teoretický základ je použit přímo v příkladech. Někdy jej lze v rámci příkladů také prohloubit. Příklady tedy slouží ke snadnějšímu pochopení teorie popř. k její prohloubení. Každý teoretický celek by měl u této metodiky obsahovat alespoň jeden příklad, který ilustruje novou teorii. Nejdélejší by bylo, kdyby se vůbec neopíral o teorii předchozích celků, pro případ, že nebyla dosatečně pochopena. To ovšem není zpravidla v praxi zcela možné. Na další příklady jež jsou součástí stejného celku (jsou-li takové nějaké), předchozí požadavek neklademe. U těch ba právě naopak není vůbec na škodu, když se opírají o starší látku a zároveň narážejí na novou. Student si pak snáze představí jejich provázanost.

V závěru každého celku by měla být uvedena cvičení. Slouží k samotnému osvojení nového, procvičení starého a nabádá studenta k aktivitě. Pokud se jedná o náročnější cvičení, měl by za zadání následovat návod k řešení a nebo by alespoň mělo být toto cvičení označeno. Některé texty určené k samostudiu cvičení vůbec neobsahují. To je podle mne velká chyba. Studenti totiž často až při řešení cvičení naráží na problémy, které si předtím ani neuvědomili a nebo je považovali za banální. Možná si autoři těchto textů myslí, že na základě teorie a řešených příkladů si student vymyslí podobný problém sám a také jej vyřeší. Neuvědomují si, že pokud čtenář z posledních sil pochopí napsané, asi stěží si bude vymýšlet cvičení, která by právě v tu chvíli tolik potřeboval.

V dalších kapitolách této diplomové práce se snažím vytvořit základní kurz JavaScriptu, jež maximálně respektuje výše navrženou metodiku.

# Úvod ke kurzu

Před několika lety znala Internet mnohem menší část populace než dnes. V roce 1992 byl vyvinut World Wide Web a ten bezesporu přispěl k dnešní popularitě Internetu. Webové stránky se pro mnoho lidí staly zdrojem informací a zábavy.

Obecně lze rozdělit webové dokumenty na statické a dynamické.

Statická stránka se prezentuje vždy stejným způsobem. Je zobrazena vždy stejná grafika, stejný text, stejné hypertextové odkazy ve stejném pořadí. Jedná se v podstatě o klasickou stránku, jejímž jediným cílem je předávat informace. Jsou dvě základní možnosti jak vytvořit statickou stránku. Buď se naučit HTML jazyk a komplet celou stránku včetně značek si napsat sám, nebo si pořídit nějaký HTML editor. V tom případě nemusíte v podstatě o HTML nic vědět. Pokud vás tvorba webových dokumentů zajme, naučíte se HTML jazyk, který není nikterak složitý. Po čase vás přestane bavit dělat statické stránky a budete chtít umět víc. Vaším dalším cílem se pak stanou pravděpodobně dynamické stránky.

Rozdíl mezi dynamickou a statickou stránkou je jako rozdíl mezi rovinou a prostorem. Dynamická stránka má mnohem víc možností než stránka statická. Lze pomocí ní informace nejen předávat, ale i získávat a vyhodnocovat. I přes sílu předchozí věty v ní nejsou obsaženy všechny možnosti dynamických stránek. Stránky libovolných vyhledávacích služeb, „chatů“, stránky s výpisem poslední aktualizace, s hodinami, s kalkulačkou, stránky s testem IQ jež vám na základě vašich odpovědí sdělí jeho hodnotu, to všechno jsou dynamické stránky. Pokud chcete vytvářet dynamické stránky stačí, když umíte používat jazyk HTML a ovládáte některý skriptovací jazyk. Nepřesně se dá říci, že pomocí HTML si vytvoříte statický dokument a pak napišete příslušný skript (kód ve skriptovacím jazyku), který vnese do vámi vytvořené stránky dynamičnost. Tento skript může pracovat jak na straně serveru tak na straně klienta. Pokud pracuje na straně serveru pak se jedná nejčastěji o CGI skript. Velká část CGI skriptů je napsána v Perlu. Pokud pracuje na straně klienta je to s velkou pravděpodobností JavaScript.

Jelikož málo kdo z vás má doma internetový server a nebo možnost na něj umístit a následně spouštět svoje skripty, dáte asi přednost psaní skriptů, jež pracují přímo na počítači uživatele (klienta). To je totiž vaše jediná rozumná a dostupná možnost tvorby dynamických stránek. V tom případě je pro vás užitečné se naučit JavaScript. A pokud přece jen můžete spouštět své skripty na serveru i vám bude jeho znalost přínosem. Pomocí JavaScriptu lze například kontrolovat data vkládaná do formulářů ještě předtím, než jsou vyplňené údaje odeslány na server. Tedy když uživatel něco napiše do formuláře, není nutné, aby byl tento formulář posílán serveru, zkонтrolován a výsledky kontroly zaslány zase zpět. Vstup je ověřen přímo klientovskou aplikací. Důsledkem toho je rychlejší odezva pro uživatele.

Tato publikace by vám měla pomoci zvládnout základy JavaScriptu. Krom nezbytné teorie obsahuje řešené příklady a cvičení. Její součástí je médium na němž najdete příklady, řešení všech cvičení a elektronickou podobu této knihy.

Od čitatele se požaduje znalost HTML jazyka (zopakujte si obzvláště tvorbu formulářů) a základní znalost algoritmizace. Základní znalost programování v některém z běžných jazyků je také téměř nutnou podmínkou pro úplné pochopení této knihy.

Pro začátek potřebujete libovolný textový editor (např. ten v kterém tvoříte běžně HTML stránky) a prohlížeč webových stránek, který umí zpracovávat JavaScript. Bohužel v době tvorby této publikace neexistoval žádný rozumný nástroj pro odladěování JavaScriptů, takže tvůrcům skriptů nezbývalo nic jiného než u každého skriptu testovat syntaktickou i sémantickou správnost opakováním spouštěním a v případě, že se vyskytla chyba (v tom lepším případě okénko jež chybu nahlásilo sdělilo i rádek, kde má být něco v nepořádku) si procítst pozorně kód skriptu, najít ji a opravit. Je celkem možné, že dnes již existuje. Pokud tomu tak je, pořídte si jej. Dále si vytvořte adresář SKRIPTY do nějž budete ukládat výsledky cvičení, nejlépe pod názvem, který najdete v závorce za zadáním. Pod stejným názvem můžete na přiloženém mediu najít moje řešení.

Při čtení vám doporučuji důkladně prostudovat příklady k teorii a svědomitě pak řešit zadaná cvičení. Jelikož funkčnost skriptů se může (ale nemusí) lišit v závislosti na operačním systému a prohlížeči, zaručuji ji plně u mych příkladů a řešení cvičení při použití operačního systému Win95 a prohlížeče Microsoft Internet Explorer 4.0 nebo Netscape Communicator 4.5. Pokud by některý příklad fungoval pouze na jednom z uvedených prohlížečů, upozorním vás na tuto skutečnost.

# Kurz JavaScriptu

Tento kurz obsahuje šest kapitol v nichž se postupně seznámíte se základy JavaScriptu.

Po přečtení první kapitoly budete umět začlenit JavaScript do HTML dokumentu.

Druhá kapitola pojednává o syntaxi JavaScriptu, jež je velmi podobná jazyku C++. Naučíte se deklarovat proměnné, funkce a seznámíte se s řídícími strukturami JavaScriptu.

Jednou z největších výhod JavaScriptu je jeho schopnost sledovat akce uživatele a přímo na ně reagovat. Akce je vlastně událost. Třetí kapitola vás seznamuje s událostmi jež lze spojit s JavaScriptem a s jejich využitím.

Ve čtvrté kapitole se dozvítíte, co to je objekt a jak tvořit vlastní objekty.

Další výuková kapitola obsahuje stručný popis objektů, které nabízí JavaScript. V rámci ní byste se s objekty neměli pouze seznámit, ale cílem je, abyste se je naučili především využívat ve svých skriptech.

Poslední kapitola je rozdělena na dvě části. První mluví o funkcích, které JavaScript zná aniž by je bylo nutné deklarovat, tzv. vestavěné funkce, a druhá podrobněji vysvětluje metodu `toString`.

Kurz je vhodné číst postupně, rozhodně ne na přeskáčku. I když některé kapitoly přímo nenavazují na předchozí, jejich řazení představuje jednu ze schůdnějších cest k zvládnutí základů JavaScriptu. Pokud některé kapitole neporozumíte, pokuste se ji pročist znova a hned si vyřešte cvičení.

## První kroky

O JavaScriptu je známé, že není přes příliš obtížný. Někdy se můžete setkat i s názorem, že se jedná o programovací nástroj pro ty, kteří neumějí programovat. Něco na tom sice je, ale pokud si uvědomíte, že neexistuje žádný rozumný nástroj pro odladování JavaScriptů, tak mnohým z vás zmizí úsměv z tváře. Právě tohle je totiž jedna z největších překážek s kterou se budete bohužel setkávat při psaní každého skriptu. V této kapitole se vám pokusím poradit, jak ji co nejelegantněji překonat a následně vám řeknu jak umísťovat skript do HTML dokumentu.

## Úprava zdrojového kódu

Při odladování vám pomůže, když je zdrojový kód vaší stránky napsán úhledně a tudíž se v něm dá i snadno orientovat. Je vhodné si zvolit (a nebo přijmout) úpravu, která vám vyhovuje, a tu pak dodržovat ve všech svých skriptech. Oceníte to při hledání chyb a také pokud budete nutenci se někdy ke svému skriptu vrátit. Co se týče HTML značek obsažených ve zdrojovém kódu stránky, i na ty se vztahuje požadavek přehlednosti. Každou značku je zvykem umístit na samostatný řádek a její název psát velkými písmeny. Nechci vám zde sáhodlouze vnucovat názor na vzhled zdrojového kódu, protože by ho stejně téměř nikdo nepřijal, ale chci zdůraznit důležitost grafické úpravy zdrojových kódů. Pokud váš skript nebude podléhat žádné cílené úpravě, na jejímž základě bude kód čitelný, riskujete, že se v něm nikdo, včetně vás, nevyzná.

Dále by měly být nezbytnou součástí kódu komentáře. U důležitých proměnných v nich popsat jejich význam, u funkcí si do nich poznamenat co mají dělat atd.. Komentář začíná dvojicí znaků /\* a končí dvojicí znaků \*/ nebo začíná znaky // a končí koncem řádku.

```
/* víceřádkový komentář
   vhodný pro popis funkčnosti celého skriptu a podobně
*/
// krátké komentáře
```

Komentáře by měly být krátké a zároveň výstižné. Jestliže vás napadá pouze dlouhý komentář veptejte jej do kódu, protože dlouhý komentář je vždy lepší než žádný.

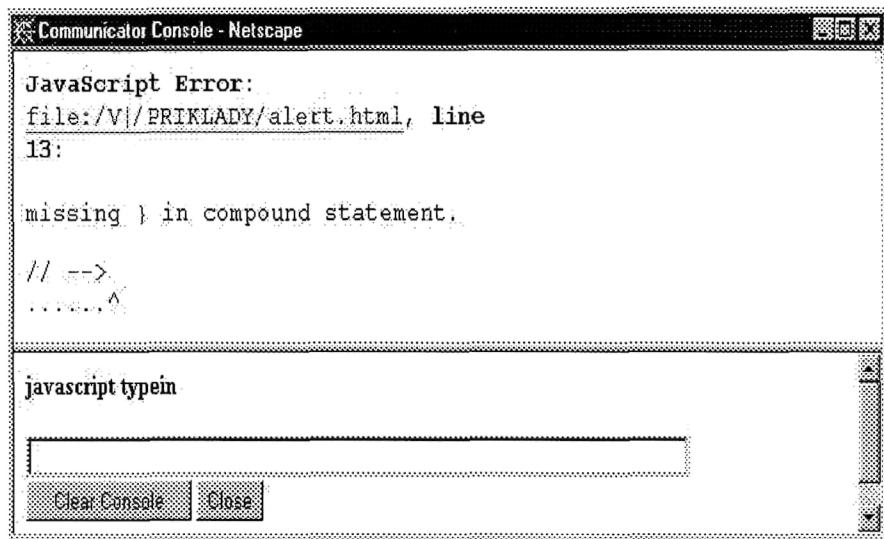
## Odladování JavaScriptů

JavaScript je jazyk skriptovací, tzn. kód je dodáván na místo spuštění ve zdrojovém tvaru, zpravidla zapsaný přímo v HTML stránce. V praxi to tedy znamená, že se spouští přímo vámi napsaný kód (není to tedy jako v Pascalu, C a mnoha dalších programovacích jazycích u nichž je nutné program kompilovat). Po vytvoření zdrojového kódu dynamické stránky, která obsahuje JavaScript, následuje bezprostředně ladění. Otevřete si novou stránku pomocí prohlížeče a hledáte její nedostatky. V případě, že byla prohlížečem objevena nějaká syntaktická chyba, zobrazí se okénko s chybovým hlášením nebo je chybové hlášení zobrazeno ve stavovém řádku prohlížeče.

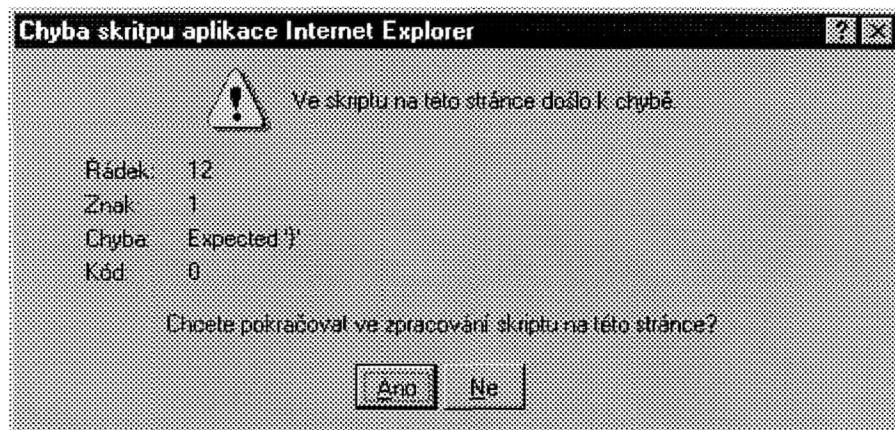
Chybové hlášení Netscapu ve stavovém řádku vypadá asi takto:



Další informace o chybě vám Netscape podá po napsání javascript: do location. Odentrování způsobí otevření JavaScript console, například:



Internet Explorer zobrazí okénko informující vás o chybě s dotazem zda chcete pokračovat ve zpracování skriptu na této stránce.

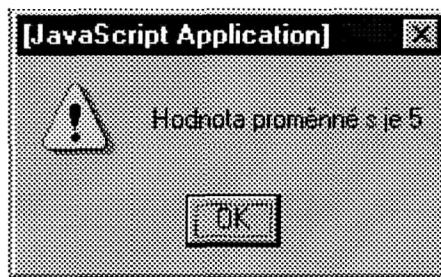


V tom lepším případě se z chybového hlášení dozvíte i řádek na kterém chyba nastala. Nalezenou chybu opravíte a nahrajete upravený dokument do prohlížeče. Pokud stránka obsahuje části kódů, které se vykonávají až na základě akce uživatele, což není u JavaScriptů nic ojedinělého, je nutné při odladění tyto kódy spustit provedením požadované akce. Když prohlížeč nevrací žádné

chybové hlášení, znamená to, že je schopen vašemu kódu porozumět a ví co má na jeho základě dělat. V této chvíli můžete říct, že váš skript je schopen nějak pracovat, ale ještě si musíte ověřit zda pracuje správně. Při srovnání s programem v Pascalu, jste se dostali do bodu, kdy se povedlo pascalovský kód zkompilovat. Správnost chodu programu v Pascalu lze ověřit s pomocí softwaru, který umožňuje sledovat hodnoty důležitých proměnných, umisťovat tzv. break pointy v nichž je program vždy pozastaven atd. Pro JavaScript žádný spolehlivý nástroj tohoto typu neexistuje a tak je nejlepší spolehnout se raději na sebe.

Pokud skript sice funguje, ale úplně jinak než by podle vás měl, pak je dobrý nápad si ověřit hodnoty důležitých proměnných. Já to dělám tak, že do vhodného místa (nejčastěji na konec) v kódu vložím příkaz alert(prom). Při každém průchodu skriptu tímto místem se zobrazí dialogové okno s hodnotou proměnné prom. Tím zjistím nejen hodnoty proměnné, kterou jsem se rozhodla takto sledovat, ale i to kolikrát skript tímto místem projde. Mnohokrát už mi to pomohlo najít problém v mém skriptu a následně jej odstranit. Používání příkazu alert(prom) při ladění vašich skriptů berte jako dobrou radu, která se vám bude v nouzi určitě hodit. Pokusím se vám ho přiblížit v několika ukázkách, aby jste mohli s jeho pomocí odhalovat chyby a protože v několika dalších příkladech a cvičeních ho budeme zneužívat k vypisování výsledku.

```
/* Proměnná s má hodnotu 5 a proměnná x má hodnotu pět. Za příslušným  
příkazem je v poznámce uveden text, který bude napsán v alert okně */  
alert(s) // 5  
alert("Hodnota proměnné s je ") // Hodnota proměnné s je  
alert("Hodnota proměnné s je " + s) // Hodnota proměnné s je 5  
alert(s + ", " + x) // 5, pět
```



Při ladění vašich skriptů vám pomůže v neposlední řadě i trpělivost. Trpělivost je totiž klíčem k mnoha dveřím. Při psaní a hledání chyb ve vašich skriptech vám ji přeje autorka.

## Začlenění skriptu do HTML dokumentu

JavaScript můžete umístit buďto přímo do zdrojového kódu dokumentu (nejčastěji do hlavičky, méně často do těla) nebo do souboru s příponou .js a v dokumentu se na něj odkázar. Do jednoho HTML dokumentu lze vložit i několik skriptů. Každý je uvozen párovou značkou SCRIPT. Její syntaxe je následující

```
<SCRIPT LANGUAGE="LangName" (SRC="URL") >  
(kód skriptu)  
</SCRIPT>
```

LangName udává jazyk použitý ve skriptovacím kódu. Ve vašem případě to bude JavaScript. Pokud je zadán atribut SRC, měl by odkazovat na adresu, obsahující soubor s příponou .js.

Velmi krátké skripty lze zapsat přímo do HTML značky na místa hodnot atributů. Nejsou uzavřeny ve značce SCRIPT.

Skripty umístějte do hlavičky dokumentu (pouze pokud k tomu budete mít opravdu pádný důvod do těla). Jelikož se zdrojový kód stránky zavádí postupně řádek po řádku od prvního až po poslední, budou nahrány jako první, což může být důležité z hlediska funkčnosti.

Některé prohlížeče neumí JavaScripty interpretovat. Jedná se buď o velmi staré prohlížeče a nebo o ty, kde to sám uživatel zakázal. Kód skriptu proto pište do HTML komentáře, aby nezpůsobil případně zmatky a byl jimi prostě ignorován. Do hlavičky dokumentu umístěte párovou značku NOSCRIPT. To co je v ní uvedeno představuje alternativní cestu právě pro tyto prohlížeče. Vepište do ní v HTML jazyce adresu dokumentu se stejnými informacemi, který JavaScript nepoužívá nebo alespoň upozornění informují o tom, že zdrojový kód obsahuje JavaScript.

Pokud se rozhodnete respektovat předchozí dva odstavečky, pak drtivá většina dokumentů jež napišete s pomocí JavaScriptu bude mít stejnou základní kostru. Ze základní kostry obyčejného HTML dokumentu vznikne vložením značek SCRIPT a NOSCRIPT do jeho hlavičky. Já jsem si ji uložila do souboru sablona.html. Prohlédněte si jeho výpis a poté si vytvořte ve svém adresáři SKRIPTY soubor se stejným názvem i obsahem.

```
Výpis sablona.html
<HTML>
<HEAD>
    <TITLE>    </TITLE>

    <SCRIPT LANGUAGE="JavaScript">
    <!--

    // -->
    </SCRIPT>

    <NOSCRIPT>
Tento dokument obsahuje kód JavaScriptu.
    </NOSCRIPT>

</HEAD>
<BODY>

</BODY>
</HTML>
```

Soubor, který jste si právě vytvořili, bohatě využijete při tvorbě vašich skriptů. Nebudete muset otrocky psát v každém kódu stále a znova tytéž značky, ale prostě si je překopírujete a nebo si otevřete soubor sablona.html ve vámi používaném textovém editoru a následně jej uložíte pod požadovaným názvem skriptu. Ušetříte tím spoustu času.

Protože kód JavaScriptu obsahuje pouze příkazy a nic víc, jste právě teď schopni vytvořit jednoduchý skript. Vyřešením následujícího cvičení se o tom přesvědčete sami.

**Cvičení:**

- 1) Vytvořte stránku, která zobrazí alert dialogové okénko s textem: „Tohle je můj první JavaScript!“. Kód JavaScriptu umístěte
  - a) přímo do zdrojového kódu HTML dokumentu. (*alert.html*)
  - b) do souboru s příponou .js. (Externí soubor .js musí vždy obsahovat pouze kód JavaScriptu!) (*prvni.html, alert.js*)

# Syntaxe JavaScriptu

Určitě umíte napsat jednoduchý program v programovacím jazyce s kterým jste se setkali. Znáte pojmy jako je proměnná, cyklus, funkce a další. I JavaScriptu jsou tyto pojmy důvěrně známé, i když někdy trošku v jiné podobě. Je tedy principiálně možné do něj přepsat algoritmy s kterými jste se setkávali, když jste začínali s programováním. Po pročtení příkladů a řešení cvičení mi dáte určitě za pravdu. Přivést vás k tomuto zjištění je ale pouze druhotný cíl této kapitoly. Jejím hlavním cílem je seznámit vás se syntaxí JavaScriptu. Striktně vzato by sem měly patřit i objekty a zejména příkazy s nimi spojené. Nezařadila jsem je na toto místo, protože o nich pojednává samostatný celek této knihy.

U všech vzorových příkladů v kapitole je uveden pouze jejich zkrácený výpis. Konkrétně je vyplán pouze samotný JavaScript, tzn. to, co je obsaženo mezi počáteční a koncovou značkou SCRIPT. Zbytek nepovažuji za nutné vypisovat, protože je totožný (když nepočítám dopsání titulku stránky) s obsahem souboru sablona.html.

## Deklarace proměnných a datové typy

Proměnná je pojmenované paměťové místo. Obsah tohoto paměťového místa je zvykem nazývat hodnotou proměnné a lze jej přepisovat. Proměnné jsou nedílnou součástí téměř veškerých skriptů a právě v tom spočívá jejich důležitost.

Názvy proměnných v JavaScriptu musí začínat písmenem nebo \_ (podtržítkem) a nesmí být rezervovaná slova. Rezervované slovo je buď klíčové slovo a nebo slovo o kterém se předpokládá, že by se v budoucnosti mohlo klíčovým stát. Jejich seznam najdete v příloze.

JavaScript je case-sensitive, to znamená, že rozlišuje malá a velká písmena. Tedy SUM, Sum a sum jsou všechno platné názvy proměnných, patřících ovšem různým proměnným. V Pascalu, který mezi velkými a malými písmeny nedělá rozdíly, by se jednalo o jednu proměnnou. JavaScript je citlivý na dodržování malých a velkých písmen i u příkazů.

Deklarace proměnných v JavaScriptu není povinná, ale nejen kvůli přehlednosti je lepší všechny proměnné deklarovat pomocí klíčového slova var. Většinou se proměnné přímo při deklaraci nastavují na smysluplnou počáteční hodnotu nebo na speciální hodnotu null. Deklarovat proměnnou lze kdekoli v programu, ale zpravidla se to dělá hned na jeho začátku. Prohlédněte si pozorně následné ukázky deklarací:

```
var x = 4          // deklarace proměnné x na počáteční hodnotu 4
var s = "23"       // deklarace proměnné s na počáteční hodnotu "23"
var u, v           // deklarace proměnných u a v, nedoporučuje se
var z = null        // deklarace proměnné z na speciální hodnotu null
```

Jak jste si určitě všimli v deklaraci není uveden ani u jedné proměnné datový typ. Mají tedy vůbec nějaký? A jaké jsou vlastně datové typy JavaScriptu?

Odpověď na první otázku je ano, ale .... Intuitivně byste jistě řekli, že x je číslo a y řetězec. Vaše intuice však není zcela správná. Zatím se s tím netrapte a věřte své intuici. Jak to opravdu je bude vysvětleno později (v části konverze typů).

Nyní se pokusím odpovědět na druhou otázku. V JavaScriptu existuje pět hlavních kategorií implicitních datových typů. Každému z nich je věnován jeden z následujících odstavečků.

Číslo (number), např. 45 nebo 5.7. V JavaScriptu existují jak desetinná tak celá čísla. Celá čísla mohou být určena ve formátu se základem 10 (desítkovém), 8 (osmičkové) a 16 (hexadecimálním). 1 až 9 následovaná jakýmkoliv souborem číslic je desítkové celé číslo, 0 následovaná libovolným souborem číslic 0 až 7 je osmičkové celé číslo, 0x nebo 0X následovaná 0 až 9, a až f nebo A až F je hexadecimální celé číslo. Při deklaraci lze tedy například psát var x = 0XA1.

Řetězec (string), např. "12" nebo 'ahoj'. Řetězce v JavaScriptu mohou být uvozeny buď v jednoduchých uvozovkách 'řetězec' nebo ve dvojitých uvozovkách "řetězec". Řetězce mohou být pomocí střídání typů uvozovek do sebe vnořeny. Speciální řetězce " nebo "" jsou řetězce nulové délky (prázdné řetězce).

Logická hodnota (boolean), true nebo false. True je „ekvivalentní“ libovolnému nenulovému číslu a false nule.

Speciální hodnota null. Má v jazyku JavaScript velmi zvláštní roli. Je to volně řečeno hodnota posledního útočiště každé proměnné. Používá se v deklaci (viz. výše). Hodnota null nepřiřazuje proměnné žádný implicitní datový typ. Může být konvertována (převedena) do vhodného tvaru předešlých typů. Jestliže je konvertována na číslo, stává se číslem 0, jestliže je konvertována na řetězec, stává se řetězcem "null" a konečně v případě, že je konvertována na logickou hodnotu, stává se hodnotou false.

Objekt (object). Objektům jsou věnovány dvě samostatné kapitoly knihy.

## Příkazy a operátory

Základní pracovní jednotka v JavaScriptu, stejně jako ve většině programovacích jazyků, je příkaz. Celý kód JavaScriptu je zpravidla složen pouze z příkazů.

Možná jste zvyklí (např. z Pascalu) příkazy ukončovat středníkem. JavaScript v tomto ohledu dává možnost volby přijetím řádkově orientovaného přístupu psaní kódu. Co to znamená? Příkaz je ukončen poté, co je dosaženo konce řádku (co rádek, to příkaz) a nebo je možné jej ukončit středníkem. Pokud jsou příkazy ukončeny středníkem, lze jich na jeden rádek napsat více. Z důvodu přehlednosti pište na jeden rádek jeden příkaz.

Mezi nejjednodušší příkazy patří přiřazovací příkaz. Jeho zápis (syntaxe) je následující:

```
x = T1
```

Jeho význam (sémantika) je stejný jako v Pascalu. Vykoná se T1 a výsledek je přiřazen proměnné x. T1 se nazývá výraz.

Výraz obsahuje proměnné, konstanty a volání funkcí, různě pospojované pomocí operátorů. Každý operátor má určitou prioritu, která rozhoduje o způsobu vyhodnocování výrazu. Pokud mají operátory shodnou prioritu, výraz se vyhodnocuje zleva doprava. Způsob vyhodnocování výrazu lze ovlivnit použitím závorek. To, co je v nich uzavřeno, se vyhodnocuje přednostně. Vzpomeňte si na matematiku, kde jste se poprvé setkali s předchozími pravidly. Např. u výrazu  $5+4*2+10$  jste nejprve vypočítali  $4*2$ , protože  $*$  má vyšší prioritu než  $+$  a až potom jste směrem zleva doprava sečetli čísla, čímž jste dostali výsledek 23. Pokud upravíme předchozí výraz přidáním závorek na výraz  $5+4*(2+10)$ , počítali by jste určitě jinak. Nejprve by jste vypočítali  $2+10$ , protože to, co je v závorce, má vždy přednost, poté vynásobili  $4*12$  a nakonec sečetli  $5+48$  a dostali výsledek 53.

Následuje tabulka operátorů JavaScriptu seřazených od nejvyšší po nejnižší prioritu. Jelikož operátory lze rozdělit na početní, logické, bitové a operátory přiřazení, je ve druhém sloupci uvedeno do které skupiny operátor patří. Třetí sloupec pak obsahuje stručný popis operátoru. Operátory, které navzájem od sebe nedělí volný řádek, mají stejnou prioritu.

Operátor	Jeho druh	Stručný popis		
<code>++</code>	početní	preinkrement, postinkrement	<code>++x, x++</code>	k x přičte 1
<code>--</code>	početní	predekrement, postdekrement	<code>--x, x--</code>	od x odečte 1
<code>!</code>	logický	logické NOT	<code>!x</code>	negace
<code>~</code>	bitový	bitové NOT	<code>~x</code>	negace bit po bitu
<code>*</code>	početní	násobení	<code>x * y</code>	
<code>/</code>	početní	dělení	<code>x / y</code>	
<code>%</code>	početní	modulo	<code>x % y</code>	zbytek po x/y
<code>+</code>	početní	sčítání, spojování řetězců	<code>x + y</code>	
<code>-</code>	početní	odečítání, unární negace	<code>x - y, -x</code>	
<code>&lt;&lt;</code>	bitový	posun do leva	<code>x &lt;&lt; konst</code>	$x * 2$ na konst
<code>&gt;&gt;</code>	bitový	posun do prava	<code>x &gt;&gt; konst</code>	$x / 2$ na konst
<code>&gt;&gt;&gt;</code>	bitový	posun do prava bez znaménka	<code>x &gt;&gt;&gt; konst</code>	$ x  / 2$ na konst
<code>&lt;</code>	logický	aritmetické a řetězcové srovnávání	<code>x &lt; y</code>	
<code>&lt;=</code>	logický	aritmetické a řetězcové srovnávání	<code>x &lt;= y</code>	
<code>&gt;</code>	logický	aritmetické a řetězcové srovnávání	<code>x &gt; y</code>	
<code>&gt;=</code>	logický	aritmetické a řetězcové srovnávání	<code>x &gt;= y</code>	
<code>==</code>	logický	rovnost	<code>x == y</code>	
<code>!=</code>	logický	nerovnost	<code>x != y</code>	
<code>&amp;</code>	bitový	bitové AND	<code>x &amp; y</code>	
<code>^</code>	bitový	XOR, tj. bitové exkluzivní OR	<code>x ^ y</code>	
<code> </code>	bitový	bitové OR	<code>x   y</code>	
<code>&amp;&amp;</code>	logický	logické AND	<code>x &amp;&amp; y</code>	

	logický	logické OR	x    y	
? :	logický	podmínkový výběr		viz. řídící struktury
= op=	přiřazení přiřazení	přiřazení souhrnné přiřazení	x = y x op= y	x = x op y
,	logický	logické spojení	x, y	

Logické operátory, kromě logického spojení a podmínkového výběru, vrací booleovskou hodnotu true nebo false podle zásad uvedených v pravdivostní tabulce, kterou jste se učili v matematice. Logické spojení se nejčastěji používá v příkazu for (viz. níže).

Bitové operátory &, |, ^ a ~ vyhodnocují vstupní hodnoty bit po bitu podle zásad uvedených v pravdivostní tabulce. XOR podle ní vrací 1, právě tehdy když jeden vstup je 1.

Přiřazení už bylo vysvětleno výše. U souhrnného přiřazení je op jeden z následujících operátorů: \*, /, %, +, -, <<, >>, >>>, &, ^, |.

Početní operátory +, -, \*, /, % mají svůj standardní význam. U operátorů ++ a -- je nutné rozlišovat prefixovou a postfixovou notaci. Podrobnější vysvětlení viz. příklad.

### Příklad:

Jakou hodnotu bude mít x, y, z a u po provedení příkazů

```
z = ++x
u = y++
```

když víme, že x = y = 10?

V prvním příkazu je nejprve vykonáno ++, takže x je 11 a potom je hodnota x přiřazena z, takže z je 11. V druhém příkazu je nejprve vykonáno přiřazení, takže u je 10 a potom ++, takže y je 11. Jedná se o praktickou ukázku rozdílu mezi preinkrementem a postinkrementem. Preinkrement nejprve změní hodnotu svého argumentu a teprve pak je vykonán příkaz či vyhodnocen výraz obsahující preinkrement. Postinkrement pracuje právě opačně, hodnotu argumentu mění až po vykonání příkazu či vyhodnocení výrazu v němž se vyskytuje. Na tento fakt si dávejte pozor, protože se jedná o jeden ze zdrojů velmi špatně odhalitelných chyb. Stejná pravidla platí i pro predekrement a postdekrement.

U příkladů, v nichž je dotaz na hodnotu výrazu nebo proměnné si můžete napsáním jednoduchého skriptu ověřit výsledek. V tomto případě by vypadal asi takto:

#### *Výpis krement.html*

```
var x = 10 // deklarace všech proměnných, vyskytujících se v příkazech
var y = 10 // x a y deklarovány na počáteční hodnotu 10 podle zadání
var z, u

z = ++x // zadaný příkaz
alert("z = " + z + ", x = " + x) // zobraz alert dialogové okénko //
// s hodnotami z a x
u = y++ // zadaný příkaz
alert("u = " + u + ", y = " + y) // zobraz alert dialogové okénko
// s hodnotami u a y
```

## Konverze typů

V části pojednávající o deklaraci proměnných jsem velmi vyhýbavě odpověděla na otázku, zda má proměnná svůj datový typ. Právě teď přišla ta správná chvíle vám říct celou pravdu.

JavaScript nemá explicitní datové typy. Není zde možnost určit, zda určitá proměnná představuje číslo, řetězec nebo logickou hodnotu. Jakákoliv proměnná může být jakýmkoliv typem. Proto je v JavaScriptu možné jedné proměnné přiřadit v deklaraci jeden datový typ a někde později v kódu jiný datový typ. Například:

```
var x = 5          // x má implicitní datový typ číslo
x = "ahoj"        // x má implicitní datový typ řetězec
```

Většina programovacích jazyků má explicitní datové typy a proto by u nich předchozí pokus skončil chybovým hlášením. V Pascalu, který má explicitní datové typy, je nutné přímo v deklaraci proměnné uvést závazně její datový typ. Proměnné lze pak přiřazovat pouze hodnoty tohoto datového typu. JavaScript má implicitní datové typy a tak nejenže nevrátí žádné chybové hlášení, ale oba příkazy provede. Tím se změní implicitní datový typ x z čísla na řetězec. Pamatujte si, že měnit implicitní typ dat je zpravidla špatné rozhodnutí i když, jak vidíte, možné.

Ale tohle ještě není celá pravda. Její zbytek se skrývá ve skutečnosti, že může být jedna proměnná v různých kontextech interpretována různě. Programovací jazyky s explicitními datovými typy tento problém nemají. Operátory mohou spojovat jen operandy stejného datového typu. Pokud mají operandy různý typ, je vráceno chybové hlášení. JavaScript oproti těmto jazykům zpracuje téměř všechno. Jestliže operátory spojují operandy stejného implicitního datového typu, které lze operátorem spojit (např. násobit dva řetězce je obecně nesmysl), vrací očekávaný výsledek a proměnná je implementovaná očekávaným způsobem. Když tomu tak není, JavaScript se pokouší hodnoty proměnných konvertovat (převést) v závislosti na kontextu tak, aby operátor spojoval hodnoty stejného typu a toto spojení bylo smysluplné. Až potom je vykonána operace. Výsledek má stejný datový typ jako obě hodnoty.

### Příklad:

```
var x = 5          // x má implicitní datový typ číslo
var y = "12"        // y má implicitní datový typ řetězec
var z = "2"          // z má implicitní datový typ řetězec
var s = "ahoj"        // s má implicitní datový typ řetězec
var b = true         // b má implicitní datový typ logická hodnota
var pom = null       // pom nemá žádný implicitní datový typ
                     // null se konverte dle pravidel uvedených výše
var v               // v nemá žádný implicitní datový typ
/* jakým způsobem budou chápány hodnoty proměnných , hodnota v včetně
typu */
v = x + y           // v = "5" + "12"      "512" řetězec (string)
v = y + x           // v = "12" + "5"      "125" řetězec (string)
v = x * y           // v = 5 * 12        60   číslo (number)
v = y * z           // v = 12 * 2        24   číslo (number)
                     // násobení řetězců není definováno
v = x * s           // pokus převést "ahoj" na číslo je neúspěšný
                     // vrátí proto NaN, bez ohledu na x bude v
                     // NaN   číslo (number)
v = z + pom         // v = "2" + "null" "2null" řetězec (string)
```

```

v = b + true      // v = 1 + 1          2      číslo (number)
v = pom + false   // v = 0 + 0          0      číslo (number)
v = !pom           // v = !false        true   logická hodnota (boolean)
v = !(-15)         // v = !true         false  logická hodnota (boolean)

```

I přesto, že jsou hodnoty proměnných dle kontextu chápány různě, jejich implicitní datový typ je stále stejný. Pouze implicitní datový typ v je stále měněn podle datového typu výsledku. Chcete-li se o tom přesvědčit, vytvořte si skript, který bude obsahovat příkazy tohoto příkladu a vložte do něj na požadovaná místa příkaz alert(typeof název\_proměnné) zobrazující datový typ proměnné.

Právě tato vlastnost JavaScriptu je zdrojem mnoha špatně odhalitelných chyb i když někdy může být i přínosem. Snažte se proto s operátory spojovat proměnné stejných typů, které jím spojovat jdou. Pokud to tak neuděláte, bude datový typ hodnoty proměnné konvertován podle kontextu. Při této konverzi používá JavaScript několik pravidel. Předchozí příklad vám některá ukázal v konkrétních případech. V následující tabulce je souhrn zobecněných pravidel. Nejedná se o jejich úplný výčet, ale pokusila jsem se zformulovat jen ty nejdůležitější, s jejichž výsledky se budete pravděpodobně setkávat nejčastěji.

Operátor/Obvyklý typ operandů	Pravidlo konverze
+ číslo, řetězec	<b>číslo, řetězec</b> – číslo se konvertuje na řetězec <b>boolean, boolean</b> – konverze na čísla (false=0, true=1) <b>řetězec, boolean</b> – boolean na řetězec <b>číslo, boolean</b> – boolean na číslo
-, *, /, %, ++, -- číslo	Jakékoli operandy použité v tomto kontextu se bude JavaScript pokoušet konvertovat na číslo. Pokud se mu to nepovede výsledek operace bude NaN. V některých případech podá JavaScript chybové hlášení. Konverze řetězce na číslo může být úspěšná pokud řetězec obsahuje pouze číslo a nebo je prázdný.
&&,   , ! logická hodnota	Jakékoli operandy použité v tomto kontextu se bude JavaScript pokoušet konvertovat na logickou hodnotu (boolean). <b>řetězec</b> – jakýkoliv neprázdný řetězec na true, prázdný řetězec na false <b>číslo</b> – jakýkoliv nenulové číslo na true, nulu nebo NaN na false

## Řídící struktury

S vašimi dosavadními znalostmi jste schopni napsat jednoduchý kód, v němž bude každý příkaz proveden právě jednou. Řídící struktury umožňují spouštět části kódu opakováně nebo jen za určitých podmínek. Pokud obsahují tyto části kódu více než jeden (případně žádný) příkaz, je nutné je uzavřít do složených závorek { }.

V JavaScriptu existují tři základní typy řídících struktur a to if, while a for. První dvě pracují zcela stejně jako if a while v Pascalu, třetí se od běžného Pascalovského for cyklu mírně liší. Ve všech figuruje podmínka. Podmínka je libovolný výraz, jehož výsledek je true nebo false a nebo číslo. Číslo 0 je ekvivalentní výsledku false, libovolné jiné výsledku true.

### Příkaz if

```
if  (podmínka)      {  
    příkazy pro if blok  
}  else  {  
    příkazy pro else blok  
}
```

Jestliže je splněna podmínka, vykonají se příkazy pro if blok, když není, vykonají se příkazy pro else blok. Část else není povinná.

#### Příklad:

Skript rozhodne o tom, zda je číslo 15 sudé nebo liché. Pomocí alert okénka zobrazí výsledek.

Výpis sudostif.html

```
var x=15
```

```
if ( x%2==0 ) // jestliže je číslo 15 dělitelné dvěma tj. 15% 2 je 0  
    alert("Číslo " + x + " je sudé. ")// zobraz alert dialogové okénko  
                                // s textem Číslo 15 je sudé.  
else           // pokud tato podmínka není splněna tj. 15%2 není 0  
    alert("Číslo " + x + " je liché. ")// zobraz alert dialogová okénko  
                                // s textem Číslo 15 je liché.
```

### Ternární operátor

```
(podmínka) ? true_příkaz : false_příkaz
```

Jestliže je splněna podmínka, vykoná se true\_příkaz, když není, vykoná se false\_příkaz. V části pro true\_příkaz, respektive false\_příkaz, může být maximálně jeden příkaz. Má stejný význam jako:

```
if (podmínka)      { true_příkaz }   else   { false_příkaz }
```

Zpravidla se dává před tímto operátorem přednost příkazu if.

#### Příklad:

Jakou hodnotu bude mít x po provedení příkazu

```
x = (y<10) ? 10 : y-10
```

když víme, že y = 100?

Jelikož podmínka  $y < 10$  není splněna, vyhodnotí se výraz  $y-10$  a jeho hodnota se přiřadí proměnné x. Hodnota proměnné x, po provedení tohoto příkazu, bude 90.

### Příkaz while (while cyklus)

```
while (podmínka) {  
    žádný nebo více příkazů while bloku  
}
```

Jestliže je splněna podmínka, vykonají se příkazy while bloku. Potom se opět vyhodnotí podmínka. Jestliže je splněna, vykonají se příkazy while bloku. Následuje opět vyhodnocení podmínky ... Tento proces se opakuje až do chvíle, kdy podmínka splněna není a nebo je cyklus přerušen speciálním příkazem (viz. níže). V tom případě se už nevykonají příkazy while bloku a pokračuje se dalším příkazem.

Pokud není záměrem vytvořit nekonečný cyklus, pak podmínka obsahuje nejméně jednu proměnnou, jejíž hodnota je měněna (aktualizována) příkazem umístěným ve while bloku nebo je ve while bloku umístěn příkaz break (viz. níže).

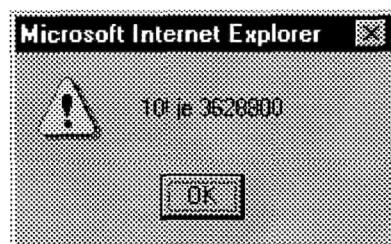
### Příklad:

Skript vypočítá 10!. Pomocí alert okénka zobrazí výsledek.

*Výpis fakwhile.html*

```
var x=1           // pomocná proměnná
var fak=1         // faktoriál

while (x<=10) {   // pokud je x menší rovno 10, tj. podmínka je splněna
    fak*=x        // fak=fak * x
    x++            // k x přičti 1, aktualizace proměnné x
}
// pokud není x<=10 ukončí while cyklus
alert(" 10! je "+fak) // zobraz alert dialogové okénko
```



### **Příkaz for (for cyklus)**

```
for (počáteční příkaz; podmínka; akt_příkaz) {
    žádný nebo více příkazů for bloku
}
```

Nejprve se provede počáteční příkaz a potom je vyhodnocena podmínka. Jestliže je splněna, vykonají se příkazy for bloku. Poté je realizován akt\_příkaz a následně vyhodnocena podmínka. Jestliže je splněna, vykonají se příkazy for bloku. Poté je realizován akt\_příkaz a následně vyhodnocena podmínka .... Tento proces se opakuje až do chvíle, kdy podmínka splněna není a nebo je cyklus přerušen speciálním příkazem (viz. níže). V tom případě se už nevykonají příkazy for bloku a pokračuje se dalším příkazem.

Počáteční příkaz je vykonán pouze jednou a to před prvním průchodem příkazy for bloku. Obvykle se používá k inicializaci proměnné. Pokud proměnnou používáme pouze uvnitř bloku, je vhodné ji v rámci počátečního příkazu i deklarovat. Zpřehledňuje to kód.

Akt\_příkaz je vykonán bezprostředně po příkazech for bloku, jako by byl umístěn za posledním příkazem v bloku for. Obvykle se používá ke změně hodnoty (k aktualizaci) proměnné, která je inicializována v počátečním příkazu.

Počáteční příkaz i akt\_příkaz může obsahovat více příkazů oddělených logickým operátorem , (čárka). Příkazy oddělené čárkou se vykonávají očekávaným způsobem jeden po druhém. Počá-

teční příkaz, podmínka i akt\_příkaz nemusí obsahovat žádný příkaz. Pokud není uvedena podmínka, tak je příkaz for vykonáván úplně stejně jako kdyby byla uvedena podmínka, která je vždy splněna. Pokud for blok neobsahuje příkaz break (viz. níže) jedná se o nekonečný cyklus.

### **Příklad:**

Skript seče čísla od 1 do 100. Pomocí alert okénka zobrazí výsledek.

*Výpis sumfor.html*

```
var suma=0 // součet
for (var x=1; x<=100; x++) // pro x od 1 do 100
    suma+=x // suma=suma + x
alert("Suma od 1 do 100 je "+suma) // zobraz alert dialogové okénko
```

Jak přesně funguje cyklus for v tomto příkladě? Nejprve je deklarovaná proměnná x na počáteční hodnotu 1. Jelikož je  $x \leq 100$ , provede se příkaz for cyklu, který k dosavadní hodnotě proměnné suma přičte x. Poté je x zvětšeno o 1 a následuje testování podmínky zda  $x \leq 100$ . Poslední dvě věty bych mohla opakovat do té doby, než x dosáhne hodnoty 101. V tu chvíli není podmínka  $x \leq 100$  splněna a skript pokračuje dalším příkazem, který zobrazí alert dialogové okénko s výsledným součtem.

## **Příkazy continue a break**

S těmito příkazy se setkáte v cyklech. Oba „násilně“ ovlivňují jejich chod.

Příkaz break (ukončí) způsobí jeho okamžité ukončení. Pokračuje se pak prvním příkazem následujícím po konci bloku cyklu. U cyklů v nichž je podmínka řídící chod cyklu vždy splněna, je to jediná možnost, jak cyklus opustit.

Příkaz continue (pokračuj) je mnohem mírumilovnější než break. Dostane-li se skript k příkazu continue, jsou všechny příkazy nacházející se mezi ním a koncem bloku cyklu, přeskočeny. U while cyklu následuje vyhodnocení podmínky, u for cyklu provedení akt\_příkazu a poté vyhodnocení podmínky. Činnost cyklu pak pokračuje očekávaným způsobem.

### **Příklad:**

Skript seče čísla od 1 do 100, která nejsou dělitelná 5. Pomocí alert okénka zobrazí výsledek.

*Výpis sumamod.html*

```
var suma=0 // součet
for (var x=1; x<=100; x++) { // pro x od 1 do 100
    if ( x%5==0 ) continue // jestliže je x dělitelné 5
                           // přeskoc následující příkazy for bloku
    suma+=x // suma=suma + x
             // je vykonáno pouze pokud x není dělitelné 5
}
alert("Suma od 1 do 100 čísel nedělitelných 5 je "+suma)
      // zobraz alert dialogové okénko
```

## Funkce

Funkce je blok kódu, který má název. Velmi často se takovému bloku kódu říká podprogram. To proto, že funkce je svým způsobem opravdu samostatný program, který lze spustit voláním v rámci skriptu obsahujícím jeho deklaraci. Deklarace funkce v JavaScriptu je následující:

```
function Název (seznam parametrů) {  
    tělo funkce  
}
```

Parametry funkce se navzájem oddělují čárkou. Seznam parametrů může být i prázdný, pak ale nezapomínejte psát závorky. Název funkce musí být jedinečný a nesmí být rezervované slovo (viz. příloha A).

Výsledek funkce je vracen pomocí příkazu `return(prom)`. Funkce v JavaScriptu nemusí vracet žádny výsledek, to když jejich hlavním posláním je vykonat příkazy v těle a nic víc se od nich nepožaduje. V Pascalu a mnoha dalších progr. jazycích se takovým funkčím říká procedury.

Funkci lze volat pomocí názvu včetně seznamu parametrů v jakémkoliv části skriptu následující po její deklaraci a nebo přímo v jejím vlastním těle či těle jiné funkce. Funkce, které volají ve svém vlastním těle samy sebe se nazývají rekurzivní. Pokud nemusíte, rekurzivní funkce nepoužívejte.

Proměnnou deklarovanou v rámci funkce nazýváme lokální. Její hodnota a i sama proměnná je známá pouze v těle této funkce. Oproti tomu proměnná, deklarovaná mimo jakoukoliv funkci, se nazývá globální a její hodnota i ona sama je známá v celém skriptu. Přesněji v jakémkoliv místě kódu, který je vykonáván za místem její deklarace.

### Příklad:

Skript zobrazí uvítací alert okénko.

*Výpis vitamvas.html*  
`function VitamVas () { /* zobrazí uvítací zprávu */  
 alert ("Vítám vás a přeji pěkný zbytek života.")  
}`

`VitamVas () // volání funkce`

### Příklad:

Skript sečte čísla od 1 do 100. Pomocí alert okénka zobrazí výsledek.

*Výpis suma.html*  
`function Suma_1_n (n) /* sečte čísla od 1 do n */  
 var suma=0 // součet  
  
 for (x=1; x<=n; x++) // pro x od 1 do n  
 // tj. pokud platí, že x<=n  
 suma+=x // suma=suma + x  
 return (suma) // předání výsledku funkce  
}  
  
var x = 0 // výsledek  
  
x = Suma_1_n (100) // x přiřadí výsledek funkce  
alert ("Suma od 1 do 100 je "+ x) // výpis výsledku`

*Výpis rek\_suma.html*

```

function Suma_1_n (n) /* sečte čísla od 1 do n, rekurzivní funkce */
    var suma=0           // součet

    if (n>0)             // podmínka zajišťující konec rekurze
        suma = n + Suma_1_n (n - 1) // rekurze
    return (suma)          // předání výsledku funkce
}

var x = 0                  // výsledek

x = Suma_1_n (100)         // x přiřadí výsledek funkce
alert ("Suma od 1 do 100 je "+ x) // výpis výsledku

```

Ve výpisu suma.html vidíte definovanou funkci s parametrem n, která pomocí příkazu return ve svém těle vraci výsledek součtu čísel od 1 do n. Za ní najdete deklaraci proměnné x. Zajímavý je následující řádek x = Suma\_1\_n (100). Suma\_1\_n (100) je volání funkce s konkrétním parametrem 100. To způsobí spuštění kódu funkce pro n rovno 100. Až je vykonán, výsledek je pomocí příkazu return předán na místo volání. To znamená, že Suma\_1\_n (100) získá hodnotu výsledku, která je vzápětí přiřazena proměnné x. A nakonec alert příkaz zobrazí požadovaný výsledek. Skript z výpisu rek\_suma.html dělá totéž, co první skript. Funkce počítající sumu je v něm však, na rozdíl od prvního skriptu, rekurzivní. Podívejme se, jak rekurze probíhá pro n=4.

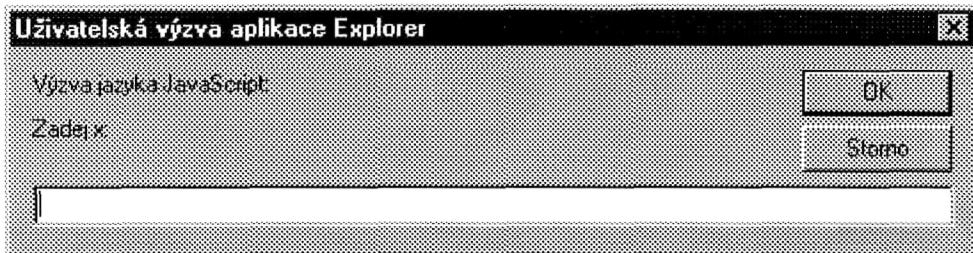
Suma_1_n (4) = 4 + Suma_1_n (3)	// n=4, n > 0 a tak rekurzivně volám Suma_1_n (3)
Suma_1_n (3) = 3 + Suma_1_n (2)	// n=3, n > 0 a tak rekurzivně volám Suma_1_n (2)
Suma_1_n (2) = 2 + Suma_1_n (1)	// n=2, n > 0 a tak rekurzivně volám Suma_1_n (1)
Suma_1_n (1) = 1 + Suma_1_n (0)	// n=1, n > 0 a tak rekurzivně volám Suma_1_n (0)
Suma_1_n (0) = 0	// n=0, neplatí, že n > 0, rekurze končí // pro n=0 vrací výsledek
Suma_1_n (1) = 1 + 0	// výpočet výsledku pro n=1
Suma_1_n (2) = 2 + 1	// výpočet výsledku pro n=2
Suma_1_n (3) = 3 + 3	// výpočet výsledku pro n=3
Suma_1_n (4) = 4 + 6	// výpočet výsledku pro n=4

Nejprve se rekurzivně zanořujeme. Až při nesplnění podmínky rekurze končí. Poté je vyhodnocena funkce pro n=0. Výsledek je předán na místo volání, což umožní vyhodnotit funkci pro n=1. Tak to pokračuje až do vyhodnocení funkce pro n=4. U rekurze si dávejte pozor, aby funkce vždy obsahovala vhodnou podmínu, která ji ukončí. Dále si pamatujte, že dobrý programátor užívá rekurzi jen tehdy, pokud k tomu má opravdu pádný důvod, jinak se jí spiše vyhýbá. I vy ji používejte zejména tehdy, jestliže vás nenapadne nebo neexistuje jiné ekvivalentní nerekurzivní řešení.

## Načítání vstupu

Logicky vzato načítání vstupu nepatří do kapitoly pojednávající o syntaxi JavaScriptu! Zařadila jsem je do jejího závěru jen proto, abyste mohli získávat vstupní hodnoty od uživatele, což vám umožní vytvářet plnohodnotnější skripty s dosavadními znalostmi. Pokusím se zde vysvětlit pouze jednu možnost, jak lze načíst vstupní hodnoty a to pomocí prompt(string, input).

Příkaz `prompt(string, input)` zobrazí okénko s nápisem `string`, přepisovatelným textovým polem o hodnotě `input`, tlačítka OK a Storno. Po zadání vstupní hodnoty do textového pole ji potvrdíme kliknutím myši na OK a nebo stornujeme výběrem tlačítka Storno. Na obrázku vidíte prompt okénko, které vznikne na základě příkazu `prompt("Zadej x:", "")`.



### Příklad:

Skript zobrazí prompt okénko. Po stisknutí tlačítka OK se objeví alert okénko s nápisem OK. Obdobně u tlačítka Storno. (U Netsape Communicatoru má stejný význam Cancel.)

*Výpis prompt.html*

```
var x=null           // vstupní hodnota

x=prompt("Zadej x:","") // zobrazí prompt
                     // při stisknutí OK x přiřadí hodnotu textového pole
if (x==null)          // jestliže je x null
  alert("Storno")      // vypíše Storno
else                  // jinak x není null, bylo stisknuto OK
  alert("OK")          // vypíše OK
```

Jak vidíte, máte možnost zjistit, zda byl vstup stornován a nebo potvrzen. Při potvrzení je řetězec napsaný v textovém poli prompt okénka přiřazen proměnné `x`. Pokud uživatel zvolí Storno, proměnné `x` je přiřazena hodnota `null`.

### Příklad:

Skript načte vstupní hodnotu a rozhodne o ní, zda je obecný řetězec a nebo řetězec obsahující číslo.

*Výpis vstup.html*

```
var x=null
var xpom=null

x=prompt("Zadej x:","")           // načte x
if ((x=="") || (x==null)) {
  alert("Nebyla zadána hodnota nebo nebyl vstup potvrzen pomocí OK.")
} else {
  xpom=1*x                         // snaha konvertovat hodnotu x na číslo
  // když byla obecný řetězec xpom je NaN a má typ number
  if ("NaN"==xpom+"") { // přičtením "" se xpom konvertuje na řetězec
    alert("Byl zadán obecný řetězec. ")
  } else {
    alert("Bylo zadáno číslo.")
  }
}
```

Skript pomocí okénka prompt načte řetězec. Jestliže byl potvrzen tlačítkem OK a je neprázdný, snaží se jeho hodnotu násobením jedničkou konvertovat na číslo. Na základě výsledku konverze, který je uložen v xpom, pak rozhodne, zda byl zadán obecný řetězec a nebo řetězec obsahující číslo. Pozorně si tento skript prostudujte. Uvidíte, že se vám to bude určitě hodit.

### Cvičení:

- 1) Jakou hodnotu budou mít následující výrazy, víte-li že  $a=2$ ,  $b=2$ ,  $c=1$ ,  $d=0$ ,  $e=4$ ? Své vlastní výpočty si zkontrolujte napsáním skriptu, který zobrazí správné výsledky. (*vyrazy.html*)
  - a)  $a++ / ++c * --e$
  - b)  $--b * c++ - a$
  - c)  $-b - --c$
  - d)  $++a - --e$
  - e)  $e / --a * b++ / c++$
  - f)  $a \%= b = d = 1 + e / 2$
  - g)  $++c * ++c/c$
- 2) S použitím ternárního operátoru napište skript, který načte dvě hodnoty a pokud to budou čísla, vrátí větší z nich. Pokud to budou řetězce, vypíše druhý v pořadí při lexikografickém uspořádání. (*vetsi.html*)
- 3) Napište skript, který načte tři hodnoty a pokud to budou čísla vrátí největší z nich. Pokud to budou řetězce vypíše poslední z nich při lexikografickém uspořádání. (*nejvetsi.html*)
- 4) Ověřte na příkladu, že podmínky if (vyraz != 0) a if (vyraz) mají totožnou funkci. Totéž ověřte i pro opačné podmínky if (vyraz == 0) a if (!vyraz). Zamyslete se nad tím, proč tomu tak je.
- 5) S použitím while cyklu napište skript, který seče čísla od 1 do 100 a pomocí alert okénka předá výsledek. (*sumwhile.html*)
- 6) S použitím for cyklu napište skript, který vypočítá  $10!$  a pomocí alert okénka předá výsledek. (*fakfor.html*)
- 7) Napište funkci IsNumber(num), která pokud je num řetězec obsahující číslo a nebo přímo číslo vrátí true jinak false. (*isnumber.js*)
- 8) Napište funkci Vstup(s), která bude zobrazovat prompt okénko tak dlouho, dokud uživatel nezadá číslo a to pak předá jako svůj výsledek. Pomocí parametru s je předáván požadovaný popis prompt okénka. (*vstup.js*)
- 9) Napište skript, který načte číslo n a pomocí alert okénka předá n!. (*faktor.html*)
- 10) Napište skript, který načte číslo n a pomocí alert okénka předá n!. n! bude počítat rekurzivní funkce. Pro  $n=4$  nebo větší se zamyslete nad tím, jak tato funkce funguje. (*rek\_fak.html*)

- 11) Napište skript, který načte 2 kladná nenulová čísla n a m a pomocí alert okénka předá
- sumu čísel od 1 do n dělitelných m. (*sumdm.html*)
  - sumu čísel od 1 do n nedělitelných m. (*sumndm.html*)
- 12) Napište skript, který načte číslo n  $2 \leq n \leq 150$  a pomocí alert okénka zobrazí prvních n členů Fibonacciho posloupnosti tj. posloupnost 1, 1, 2, 3, 5, 8, atd.. Každý další člen vznikne součtem předchozích dvou. (*fib.html*)
- 13) Napište skript, který načte číslo n  $1 \leq n \leq 150$  a pomocí alert okénka zobrazí n-tý člen Fibonacciho posloupnosti. (*fibclen.html*) Promyslete si, jak by vypadala rekurzivní funkce pro výpočet n-tého Fib. členu. Pro n=4 nebo větší se zamyslete nad tím, jak by fungovala.

# Události a JavaScript

JavaScript má, narozdíl od Pascalu, možnost reagovat přímo na některé akce uživatele nazývané v této statii události. Zpravidla je na základě události spuštěna funkce JavaScriptu. Jak toho docílit? Které události JavaScript zná? Nejen na tyto otázky najdete odpovědi v následujícím textu.

## Přehled událostí JavaScriptu

Událost	Popis
<b>onLoad</b>	Událost je vyvolána po natažení dokumentu do okna prohlížeče nebo do všech rámů v rámci jednoho FRAMESET. Atribut může být použit u elementu BODY a FRAMESET.
<b>onUnLoad</b>	Událost je vyvolána po odstranění dokumentu z okna nebo rámu. Atribut může být použit s elementy BODY a FRAMESET.
<b>onClick</b>	Událost je vyvolána po kliknutí myší na element. Atribut může být použit u většiny elementů.
<b>onDbClick</b>	Událost je vyvolána po dvojitém kliknutí myší na element. Atribut může být použit u většiny elementů.
<b>onMouseDown</b>	Událost je vyvolána po stisknutí tlačítka myši nad elementem. Atribut může být použit u většiny elementů.
<b>onMouseUp</b>	Událost je vyvolána po uvolnění tlačítka myši nad elementem. Atribut může být použit u většiny elementů.
<b>onMouseOver</b>	Událost je vyvolána při přesunutí myši nad elementem. Atribut může být použit u většiny elementů.
<b>onMouseMove</b>	Událost je vyvolána při pohybu myši nad elementem. Atribut může být použit u většiny elementů.
<b>onMouseOut</b>	Událost je vyvolána po odsunutí myši z elementu. Atribut může být použit u většiny elementů.
<b>onFocus</b>	Událost je vyvolána v okamžiku, kdy je element aktivován myší nebo pomocí tabulátoru. Atribut je možno použít u elementů LABEL, INPUT, SELECT, TEXTAREA a BUTTON.
<b>onBlur</b>	Událost je vyvolána v okamžiku, kdy element přestává být aktivní. Atribut je možno použít u elementů LABEL, INPUT, SELECT, TEXTAREA a BUTTON.
<b>onKeyPress</b>	Událost je vyvolána po stisku a následném uvolnění tlačítka na klávesnici. Atribut může být použit u většiny elementů.
<b>onKeyDown</b>	Událost je vyvolána po stisku tlačítka na klávesnici. Atribut může být použit u většiny elementů.
<b>onKeyUp</b>	Událost je vyvolána po uvolnění tlačítka na klávesnici. Atribut může být použit u většiny elementů.
<b>onSubmit</b>	Událost je vyvolána při odesílání formuláře. Atribut může být použit pouze u elementu FORM.
<b>onReset</b>	Událost je vyvolána po vynulování formuláře. Atribut může být použit pouze u elementu FORM.
<b>onSelect</b>	Událost je vyvolána po označení textu ve vstupním poli. Atribut může být použit u elementů INPUT a TEXTAREA.
<b>onChange</b>	Událost je vyvolána pokud se změnila hodnota vstupního pole formuláře. Atribut může být použit u elementů INPUT, SELECT a TEXTAREA.

## Začlenění událostí do dokumentu

Z předchozí tabulky jste se dozvěděli, které události může JavaScript využívat. Druhý sloupec obsahuje nejen jejich stručný slovní popis, ale i informaci o tom u kterého HTML elementu, jimižmi slovy u které HTML značky, lze událost použít. Události se totiž vždy vztahují k nějakému elementu. Jeden element může obsluhovat i několik událostí. Do dokumentu je začleňujeme takto:

```
<HTMLznačka NázevUdálosti = "kódJavaScript">
```

Jak vidíte, vypadají zcela stejně jako obyčejné atributy HTML k nimž je přiřazen kód JavaScriptu. Kód může být psán doslovně, může odkazovat na funkci (funkce) skriptu nebo může být kombinací obou uvedených možností. Obecně se snažíme, aby nebyl moc dlouhý, takže dáváme přednost volání funkcí.

Kód skriptu spojený s událostí je vykonán až když tato událost nastane. Pokud událost nikdy nenastane, nikdy se nespustí s ní spojený kód a tudíž nelze v danou chvíli nic tvrdit o jeho správnosti. Možná v sobě skrývá záladné chyby, možná je nesmyslný a v neposlední řadě je možná i správný, kdo ví. Při odladěvání je proto nutné odzkoušet postupně všechny události nad danými elementy, které jste deklarovali. Jen tak se přesvědčíte, že váš skript funguje opravdu tak, jak má.

### Příklad:

Skript umožňuje změnit barvu pozadí dokumentu zaškrtnutím výběrového radio „tlačítka“. Při načtení dokumentu je o tom uživatel informován a při odstranění dokumentu je mu poděkováno za návštěvu.

```
Výpis bgfocus.html
<HTML>
<HEAD>
    <TITLE> Události - změna barvy pozadí </TITLE>

    <SCRIPT LANGUAGE="JavaScript">
    <!--

        function Upozorni() {
            alert("Barvu pozadí lze změnit zaškrtnutím radio tlačítka")
        }

        function BarvaPozadi(color) { //změní barvu pozadí
            document.bgColor= color;
        }

    // -->
    </SCRIPT>

    <NOSCRIPT>
Tento dokument obsahuje kód JavaScriptu.
    </NOSCRIPT>

    </HEAD>
    <BODY bgColor="#000000"
          onLoad="Upozorni()" onUnLoad="alert('Díky za návštěvu. ')">

    <FORM>
        <INPUT TYPE=radio NAME="barva" VALUE="#000000" checked
```

```

        onFocus="BarvaPozadi ('#000000') ">
<FONT COLOR="#000000"> černá </FONT>
<INPUT TYPE=radio NAME="barva" VALUE="#ff0000"
        onFocus="BarvaPozadi ('#ff0000') ">
        <FONT COLOR="#ff0000"> červená </FONT>
<INPUT TYPE=radio NAME="barva" VALUE="#0000ff"
        onFocus="BarvaPozadi ('#0000ff') ">
        <FONT COLOR="#0000ff"> modrá </FONT>
</FORM>

</BODY>
</HTML>

```

S HTML značkou BODY jsou v tomto příkladě spojeny hned dvě události. První onLoad volá funkci Upozorni() při načtení dokumentu, která zobrazí alert okénko. Druhá onUnLoad zobrazí alert okénko, děkující za návštěvu, když uživatel opouští stránku. Kód JavaScriptu je v tomto případě uveden rovnou u události. Kdyby se jednalo o víc příkazů, musely by být buď každý na novém řádku a nebo odděleny středníky, tak jak jste zvyklí u všech kódů JavaScriptu. Dále si všimněte použitých uvozovek. Samotný kód JavaScriptu je uzavřen v dvojitých uvozovkách. Argument příkazu alert je ohrazen jednoduchými uvozovkami. Je to nutné, protože se jedná vlastně o vnořený řetězec do řetězce kódu JavaScriptu. Následující HTML značky definují formulář s radio vstupním polem, které umožňuje zvolit právě jednu možnost z nabízených. Při zatřžení požadované barvy je spuštěna funkce měnící pozadí. To je způsobeno událostí onFocus. Úplně stejného výsledku lze dosáhnout použitím události onClick místo onFocus. Přesvědčete se o tom sami.

### Cvičení:

- 1) Odpovězte na otázky:
  - a) Co to je událost? Uveďte příklad.
  - b) Kdy se vykoná kód JavaScriptu spojený s událostí?
  - c) Můžete libovolnou HTML značku (HTML element) ošetřit libovolnou událostí? Při odpovědi ne uveďte alespoň jeden protipříklad.
  - d) Má smysl jednu HTML značku ošetřit událostmi onDoubleClick a onClick? Svoji odpověď zdůvodněte.
- 2) Napište skript, který po kliknutí myši na tlačítko (button) s nápisem n!, načte číslo n a pomocí alert okénka předá n!. (*ufaktor.html*)
- 3) Při natažení dokumentu s tlačítky inkrement a dekrement načtěte číslo n. Po kliknutí na tlačítko inkrement, resp. dekrement, přiřaďte n inkrement, resp. dekrement, n a zobrazte pomocí alert okénka jak starou tak novou hodnotu n. Např. pokud n=5, kliknutí na inkrement zobrazí alert s textem "inkrement 5 je 6". Další kliknutí na inkrement zobrazí alert s textem "inkrement 6 je 7". Uchováváte číslo n jako globální nebo jako lokální proměnnou? Svoji odpověď zdůvodněte. (*krement.html*)

- 4) Vytvořte stránku s výběrovým polem (select) o položkách suma, faktoriál a tlačítkem s nápisem Počítej. Kliknutí myši na tlačítko způsobí načtení čísla n, vypočtení sumy od 1 do n, pokud je vybrána suma resp. vypočtení n!, pokud je zvolena položka faktoriál. Výsledek výpočtu zobrazte pomocí alert okénka. (*uselect.html*)
- 5) Vytvořte stránku s černým, červeným a modrým jednobarevným obrázkem. (Najdete je v podadresáři OBRAZKY adresáře CVICENI.) Odsunutí myši z obrázku změní barvu pozadí na barvu obrázku. Dvojité kliknutí myši nad obrázkem zobrazí hexadecimální kód nebo anglický název jeho barvy pomocí alert okénka. Pokud uživatel stiskne tlačítko na klávesnici, objeví se alert okénko s návodem, jak lze pracovat s tímto skriptem. (*ubgobr.html* jen IE)

# Objekty

O JavaScriptu se někdy tvrdí, že je to jazyk postavený na objektech. Mnozí z vás se však s tímto pojmem ještě nesetkali a když tak letmo. Je to dáno tím, že tvorba objektů se neprobírá v základních kurzech Pascalu či C++, i když se v těchto programovacích jazycích objekty tvořit dají. Tato kapitola by vám měla pomoci pochopit, co to objekt je a nastinit jaké možnosti přináší. Dále vám vysvětlím, jak jej lze vytvořit a jak s ním můžete následně pracovat. Čtěte, prosím, pozorně nejen proto, že tato kapitola bude pro některé asi obtížnější než ty předchozí, jelikož pojednává o něčem novém zcela neznámém, ale i proto, že na ni volně navazuje kapitola následující. Pokud se však důkladně soustředíte, jistě se vám ji povede nastudovat. Možná vás pak objekty oslovi natolik, že už si bez nich nebudeš umět představit plnohodnotné programování. Kdo ví.

## Úvod do objektů

Objekt je vlastně pouze jiný, pro vás asi nový, pohled na data. Doposud většina z vás při tvorbě skriptů přemýšlela více nad příkazy a potřebná data byla spíše podružná. Proč se jimi také zabývat. Kdykoliv je lze dodeklarovat a navíc nejsou rozhodně hlavou programu. Zkusme společně obrátit dosavadní postup návrhu algoritmu a začneme od dat. Data určitým způsobem strukturalizujeme a budeme je nazývat objekty. Dále svůj pohled zásadně změníme v tom, že nebudeme přemýšlet o tom, co budeme dělat s objekty my, ale o tom, co nám budou nabízet, tj. jaké akce budou uskutečňovat. Tyto akce představují jejich schopnosti (metody). Každý objekt charakterizuje nejen jeho schopnosti, ale i vlastnosti. Základem programu při tomto pohledu jsou data, která něco umí a mají nějaké vlastnosti - objekty. Programování v tomto duchu se pak nazývá objektově orientované programování. Mezi největší výhody takového přístupu patří snadnější rozšiřitelnost a čitelnost programu. Navíc lze vhodně navržené objekty uložit do samostatných souborů s příponou .js a tím si vytvořit přehlednou knihovničku objektů, která se vám bude určitě jednou hodit. Objektově orientované programování je tedy nový způsob myšlení postavený na datech majících schopnosti a vlastnosti. Některé objekty mohou mít jen vlastnosti či schopnosti, jiné obojí. Rozdíl mezi vlastností a schopností se pokusím přiblížit na příkladech.

Prvním objektem našeho zájmu bude člověk (třeba sousedova dcera a nebo syn). Vlastnosti tohoto objektu jsou např. barva vlasů, barva očí, výška, váha atd. Schopnosti je např. přesunout se na povel z místa na místo, uvařit oběd z předložených surovin dle receptu, zašít kalhoty, opravit auto atd.

Další objekt bude obdélník. Mezi jeho vlastnosti budou určitě patřit rozměry a, b. Schopnostmi by mohl být výpočet plochy, obvodu.

Jak jste mohli z předchozích dvou příkladů vytušit, vlastnosti jsou zpravidla konstanty resp. promenné, kdežto schopnosti představují funkce. Funkce spojené s metodami objektů mohou, ale

nemusí, vracet výstupní hodnotu, mohou, ale nemusí, mít parametry. To vše závisí na tom, co má daná metoda umět a na vašem přístupu.

Sami si vymyslete další objekt a zamyslete se nad jeho vlastnostmi a metodami.

## Deklarace a tvorba objektů

Co znamená slovo objekt již víte. Je to datová struktura skládající se z vlastností a metod. Pokud nalistujete o několik stránek dopředu, uvědomíte si, že je to i datový typ zrovna tak, jako např. číslo či logická hodnota. Mezi datovými typy však zaujímá zvláštní postavení. Objekt se musí nejprve deklarovat, tzn. vytvořit datovou strukturu s námi zvoleným názvem. Až poté lze deklarovat proměnné s touto strukturou obsahující konkrétní hodnoty vlastnosti. Místo o proměnné typu objekt je zvykem mluvit o instanci objektu. U všech jiných datových typů je nutné deklarovat pouze proměnné. Dále chcete-li např. změnit hodnotu některé proměnné, stačí k tomu název proměnné a obyčejný přiřazovací příkaz. Instance objektů jsou v tomto trochu složitější. Chcete-li změnit hodnotu některé vlastnosti s názvem instance rozhodně nevystačíte. Co ted?

Cílem této statí je naučit vás deklarovat nové objekty, tvořit jejich instance a pracovat s nimi.

### Deklarace

Objekty se v JavaScriptu deklarují definováním zvláštního druhu funkcí. Při deklaraci veškerých vlastností a metod se používá velmi důležité klíčové slovo `this`, které vždy odkazuje na aktuální objekt. V tomto případě ukazuje na vytvářený objekt. Samotná deklarace se dá popsat asi takto:

```
function NázevObjektu (vlastnost1, ..., vlastnostN) {
    this.v1=vlastnost1
    ...
    this.vN=vlastnostN
    this.metodal=Funkce1 // popř. this.metodal=Funkce1(parametry)
    ...
    this.metodaN=FunkceN // popř. this.metodaN=FunkceN(parametry)
}

function Funkce1 (parametry) {
    kód funkce
}
...
function FunkceN (parametry) {
    kód funkce
}
```

Nejprve je definován funkci `NázevObjektu` popis nového objektu obsahující všechny jeho vlastnosti a metody. Jak sami vidíte, vlastnosti jsou parametry této funkce. Právě to jim umožňuje přiřadit hodnotu při tvorbě konkrétní instance (proměnné) objektu. Kdyby měly mít všechny instance některou vlastnost stejnou (konstantní), označme ji `vk`, nebyla by uvedena mezi parametry. V těle funkce `NázevObjektu` by byla deklarována jako první (jediným důvodem je přehlednost) příkazem `this.vk=konstanta` a až pak by následovaly příkazy tvořící nekonstantní vlastnosti a metody objektu. U metod je důležité, aby funkce spojená s tou kterou metodou byla řádně definována (ve stejném dokumentu jako funkce `NázevObjektu`). Objekt lze obohatit o metodu příkazem

určeným právě k tomuto účelu `this.metoda=Funkce`, v opravdu krajním a opodstatněném případě příkazem `this.metoda=Funkce(parametry)`.

### Příklad:

Objekt popisující obdélník, který má vlastnosti rozměry a, b a metody výpočet plochy a obvodu.

#### *Výpis obd1.js*

```
function Obdelnik(ma,mb) { /* definice objektu Obdelnik
                           ma a mb jsou jeho rozměry */
    this.a=ma           // vlastnost rozměr a
    this.b=mb           // vlastnost rozměr b
    this.o=ObvodObd(ma,mb) // metoda pro výpočet obvodu obdélníku
    this.S=PovrchObd(ma,mb) // metoda pro výpočet obsahu obdélníku
}
/* funkce spojené s metodami */
function ObvodObd(a,b) { /* vrací obvod obdélníku */
    var o=0
    o=2*(a+b)
    return(o)
}
function PovrchObd(a,b) { /* vrací povrch obdélníku */
    var s=0
    s=a*b
    return(s)
}
```

#### *Výpis obd2.js*

```
function OObdelnik(ma,mb) { /* definice objektu OObdelnik
                           ma a mb jsou jeho rozměry */
    this.a=ma           // vlastnost rozměr a
    this.b=mb           // vlastnost rozměr b
    this.o=ObvodObd    // metoda pro výpočet obvodu obdélníku
    this.S=PovrchObd  // metoda pro výpočet obsahu obdélníku
}
/* funkce spojené s metodami */
function ObvodObd() { /* vrací obvod obdélníku */
    var o=0
    // this.a - hodnota vlastnosti a objektu, jež metodu volal
    o=2*(this.a+this.b) // this.b - hodnota vl. b volajícího objektu
    return(o)
}
function PovrchObd() { /* vrací povrch obdélníku */
    var s=0
    // this.a - hodnota vlastnosti a objektu, jež metodu volal
    s=this.a*this.b    // this.b - hodnota vl. b volajícího objektu
    return(s)
}
```

Vytvořila jsem hned dva objekty, které splňují zadání. Oba dva mají stejnou strukturu. Nejprve definice samotného objektu, ve které si všimněte klíčového slova `this`, jež je nutné použít s každou novou položkou. Poté vlastní funkce spojené s metodami výpočtu obvodu a obsahu obdélníku. Podíváte-li se však na oba výpisy podrobněji, najdete odlišnosti.

Objekt `Obdelnik` je totiž „méně objektový“ než `OObdelnik`. Jeho funkce spojené s metodami by šly použít i samostatně. Byly vytvořeny nezávisle na objektu `Obdelnik` a tudíž je nutné jim předat

rozměry obdélníku pomocí parametrů. Jelikož jsou jejich hodnoty známé a to konkrétně ma, mb (resp. this.a, this.b), můžeme (nebo spíš musíme) toho využít při definici metod. Proto obsahuje přiřazení funkce k metodě i parametry s hodnotami ma, mb. Metoda v tomto pojetí není striktně vzato metodou, i když se tak jeví. Deklarační příkaz this.o=ObvodObd(ma,mb) totiž neznamená nic jiného než přiřazení výsledku funkce do this.o. Stejně tvrzení platí pro deklaraci metody počítající obsah obdélníku. Takže S nebo o jsou vlastně pouze místa na odkládání výsledků. Nešlo by to udělat jinak, lépe? Je nutné předávat rozměry obdélníku pomocí parametrů, když jsou vlastnostmi objektu? Jedná se určitě o nejlepší řešení? Tento přístup opravdu není tím nejšťastnějším, i když je to také možnost.

JavaScript nabízí ještě jedno, mnohem elegantnější, řešení. Prohlédněte si výpis souboru obd2.js, kde je deklarován objekt OObdelnik. Zde jsou funkce ObvodObd() a PovrchObd() pevně spjaty s objektem. Rozměry obdélníku se v nich získávají přímo z vlastnosti objektu pomocí klíčového slova this (nikoliv z parametrů). Délku strany a vrátí this.a, obdobně strany b this.b. Deklarační funkce OObdelnik má, stejně jako Obdelnik, parametry ma, mb, které představují míry obdélníku. Obsahuje také definici vlastností pojmenovaných a a b i metod o a S. Všimněte si ale, že definice metod je odlišná. Metodě je přiřazen pouze název funkce bez závorek! Připsáním závorek by se nám metody zhroutily. Např. příkaz this.o=ObvodObd() by totiž, jako u prvního objektu, znamenal přířadit výsledek funkce. Funkce by ale vůbec nic netušila o tom, že je volána z objektu. Jak by to také měla vědět. V důsledku toho by this.a i this.b obsažené v těle funkce nebyly známé. Právě to by způsobilo problémy. Pokud u názvu funkce závorky nejsou připsány, je „volání metod ztotožněno s voláním funkce“ a lze se v těle funkce bez problémů odkazovat na vlastnosti objektu, zpravidla pomocí klíčového slova this.

Pamatujte si, že při tvorbě nového objektu je nejvhodnější si nejprve rozmyslet, jaké bude mít vlastnosti a metody, na základě toho jej deklarovat a až poté dopsat samotné funkce vztahující se k metodám. Funkce by neměly mít mezi parametry vlastnosti objektu (jiné parametry mít samozřejmě mohou), protože je není problém získat s použitím klíčového slova this. Pouze takto vytvořené objekty lze považovat za plnohodnotné objekty. Objekt OObdelnik je ukázkou toho, jak to může vypadat. Objekt Obdelnik je spíše ukázkou odstrašujícího příkladu, kvůli pofiderní definici jeho metod. Pokud k tomu nebudeš donuceni, neříďte se jím. I když jen tak mezi námi, někdy se stává, že někde objevíte funkci, která se vám hodí zrovna jako metoda k objektu, jež právě tvoříte. Bojíte se však do ní, jak už to tak bývá, jakkoliv zasahovat. V tom případě je pro vás asi nejvhodnější si vzít objekt Obdelnik za vzor.

## Tvorba a používání instancí

V této chvíli máte představu, jak vybudovat nový objekt. Je to něco jako mít plány na dům. Podle plánů se dá postavit libovolný počet konkrétních domů. Ale i když budou mít všechny stejné místnosti, budou se rozhodně lišit adresou a dá se předpokládat, že i nábytkem. V řeči objektů místo stavění domů tvoříme konkrétní instance, místo o adrese domu mluvíme o názvu instance a místo nábytku má objekt hodnoty vlastností. Zrovna tak jako nábytek je můžeme měnit s ohledem na předpokládaný typ dat. Např. bude-li mít objekt vlastnost délka, nebylo by asi příliš účelné ji

přiřadit hodnotu true či nějaký řetězec, přestože to není nemožné. Ostatně do obývacího pokoje si také nikdo nedává kuchyňskou linku – leda tak blázen.

Pokud chceme deklarovaný objekt použít, je nutné vytvořit alespoň jeden jeho konkrétní případ. V přirovnání s domy to znamená postavit nový dům a nastěhovat do něj nábytek. To se dělá pomocí příkazu new ve spojení s vyvoláním funkce:

```
var MyObjekt = new NázevObjektu(hodnotav1, ..., hodnotavN)
```

Vytvořená instance má název MyObjekt s hodnotami vlastností, které jsou předány pomocí parametrů funkce NázevObjektu. Samozřejmě má k dispozici nejen všechny vlastnosti, ale i metody patřící obecnému objektu.

Pro snadnější pochopení bude nejhodnější se vrátit k objektům z příkladu a ukázat si to na nich.

```
// instance obdelnik1 objektu Obdelnik  
var obdelnik1 = new Obdelnik(5,8)  
                    // instance obdelnik2 objektu OObdelnik  
var obdelnik2 = new OObdelnik(2,4)
```

Předchozí příkazy deklarovaly dvě instance, od každého objektu jednu. Všimněte si, že pomocí parametrů jim byly předány konkrétní hodnoty vlastností. Např. obdelnik2 má rozměry 2 a 4 jednotky, tj. vlastnost a má hodnotu 2, b 4. Obdobně obdelnik1.

Nyní, když jsou vytvořeny instance obdelnik1 a obdelnik2, vás zřejmě napadá otázka, jak se odkazovat na jejich vlastnosti či metody. Odpověď je hned několik.

První už mnozí z vás určitě tuší. Stačí použít operátora tečka. Naše objekty mají vlastnosti a a b. obdelnik1.a obsahuje číslo 5, obdelnik1.b číslo 8, obdelnik2.a číslo 2 a obdelnik2.b číslo 4. Takže stačí napsat název instance tečka vlastnost. Při volání metod je postup téměř totožný. Konkrétně u instance objektu Obdelnik používáme příkaz obdelnik1.o, resp. obdelnik1.S, avšak u instance objektu OObdelnik obdelnik2.o(), resp. obdelnik2.S(). Jestliže by funkce spojená s metodou měla parametry, nesměly by v závorkách chybět jejich hodnoty. Zda je nutné při volání metod použít závorky (včetně případných parametrů) nebo ne, je přímo závislé na definici objektu. Pokud si vzpomínáte, tak při tvorbě metod u objektu Obdelnik byly funkce metod přímo volány včetně parametrů. Na tomto místě z tohoto důvodu nemohou být závorky uvedeny. Vznikl by syntaktický nesmysl. Naproti tomu u objektu OObdelnik bylo pouze pomocí názvu odkazováno na patřící funkci, proto zde nesmí závorky chybět. Takže pokud použijete závorky (včetně parametrů) u tvorby metod, při volání metod nesmí být. Naopak pokud je nepoužijete, při volání metod ne smí chybět.

Jelikož se dá předpokládat, že většinu svých objektů budete vytvářet způsobem, který naznačuje objekt OObdelnik, nadále se budu zabývat pouze jimi.

V předchozích řádcích jsem vám ukázala, jak se můžete na vlastnosti a metody objektů odkazovat pomocí operátora tečka. Existuje však další možnost, která využívá polí. Běžné programovací jazyky podporují tento datový typ. Pole je množina oindexovaných prvků. Prvek s indexem n získáte pomocí příkazu NázevPole[n]. Pod pojmem index se vám asi automaticky vybaví číslo.

Indexem však obecně nemusí být pouze čísla. Pole, jejichž indexy nejsou čísla, nazýváme asocia-tivní. Proč mluvíme o asocia-tivních polích a ne o objektech? V JavaScriptu je pole a objekt jen rozdílný pohled na tutéž věc. Každé pole je objekt (proto o něm budu mluvit až v rámci objektů JavaScriptu) a objekt zase pole. Objekty lze tedy chápát jako asocia-tivní pole. Instance obdelnik2 je z tohoto pohledu asocia-tivní pole. obdelnik2['a'] má hodnotu 2, obdelnik2['b'] 4. Pokud chceme přes asocia-tivní pole používat i metody, musíme připsat závorky a vše pak bude vypadat takto: obdelnik2['o'](), resp. obdelnik2['S'](). Jen tak pro zajímavost si zkuste pomocí alert okénka zobrazit hodnotu prvku obdelnik2['o'] nebo obdelnik2['S'] a zamyslet se nadní.

Není opravdu až tak běžné volat přes asocia-tivní pole metody objektu, ale obsahuje-li objekt pouze vlastnosti, může být někdy mnohem vhodnější ho chápát jako asocia-tivní pole. A naopak chceme-li vytvořit asocia-tivní pole, jedna z možností je vhodně definovat objekt (další možnost viz. Array). Jeho výhody oceníte pokud si budete chtít vytvořit jednoduchý slovníček, telefonní seznam atd.

### Příklad:

Jednoduchý telefonní seznam. Pokud neobsahuje telefonní číslo zadанé osoby, zobrazí stručný návod, jak je do seznamu zařadit. Jména osob musí být bez háčků a čárek.

*Výpis seznam.html (pouze samotný kód JavaScriptu)*

```
function Seznam() { /* objekt telefonní seznam */
    this.Martina = "0501/494295" // název vlastnosti je jméno
    this.Lenka = "02/334578" // hodnota telefon
    this.Lee = "0501/354266"
}

var seznam = new Seznam() // vytvoření instance objektu Seznam
// aktivace Seznamu
var x="" // jméno osoby
var cislo="" // telefonní číslo

x=prompt("Jméno osoby:","") // načti jméno osoby
cislo=seznam[x] // její telefon vlož do prom. cislo

if (typeof cislo=="string") { // pokud je typ proměnné cislo string
    // tj. osoba existuje
    alert(cislo) // vypiš její telefonní číslo
} else {
    // pokud osoba neexistuje
    // (typeof cislo je undefined) vypiš
    alert("Až vám dá "+x+" telefon, dopište do funkce Seznam řádek
this."+x+"=telefon") // ve skriptu na jednom řádku
}
```

Objekt Seznam je pěkná ukázka běžného asocia-tivního pole. Obsahuje pouze vlastnosti. Název vlastnosti je vždy jméno osoby a hodnota řetězec obsahující její telefonní číslo. Za deklarací seznamu je vytvořena jeho instance. Samotný skript při načtení zobrazí prompt okénko. Do něj se zadává jméno osoby, jejíž telefon nás zajímá. To je přiřazeno proměnné x. Když je osoba zařazena v seznamu, pak seznam[x] vrátí příslušné telefonní číslo. Pokud ne, seznam[x] není definován a jeho typ je undefined. Co se stalo? Zajímala nás hodnota neexistující vlastnosti. Např. zadáme-li jméno Franta, ptáme se po hodnotě vlastnosti s názvem Franta. Jenže vlastnost Franta není

definována a tím spíš ani její hodnota, kterou vrací příkaz `seznam["Franta"]` nebo `seznam.Franta`. Proto je `seznam[x]` ne definován typu `undefined`. Proměnné `cislo` je přiřazen v každém případě výsledek příkazu `seznam[x]` a to včetně typu. Hned v dalším řádku kódu jsem toho využila. Jestliže má `cislo` datový typ `string`, telefon byl nalezen a následně jej pomocí `alert` okénka vypisuji. Jinak je osoba neznámá, `cislo` nemá datový typ `string`, a tak vypisuji návod jak o ni `seznam` rozšířit. Jinou možností zařazení osoby do telefonního `seznamu`, je v těle skriptu uvést příkaz `seznam["osoba"]="telefon"` popř. `seznam.osoba="telefon"`. Vyplývá z faktu, že instance objektů a tudíž i pole jsou dynamicky rozšiřitelné struktury. Napišete-li tedy např. za deklarací proměnných `x` a `cislo` řádek `seznam["Franta"]="0501/963254"` bude to mít podobný efekt, jako kdyby jste přímo do funkce `Seznam` vložili `this.Franta="0501/963254"`. Proč podobný a ne stejný? Protože příkaz `seznam["osoba"]="telefon"` rozšiřuje pouze instanci `seznam` a ne objekt `Seznam`. Přesvědčit se o tom dá tak, že si vytvoříte ještě jednu instanci. Takto dodefinovaná osoba ji nebude známa. Oproti tomu dodefinováním osoby přímo ve funkci `Seznam`, změníme objekt, základ instancí. Je tedy logické, že všechny instance budou tuto osobu znát.

## Vnořené objekty

Vlastnost objektu může být konstanta nebo proměnná jakéhokoliv implicitního typu. Každý objekt je nový datový typ a každá instance svého objektu je nová proměnná, která má svůj objekt jako implicitní typ `instance`. Z těchto dvou vět plyne, že objekty v sobě mohou obsahovat jiné objekty. Stačí vlastnosti přiřadit proměnnou typu objekt. To je v mnoha případech velmi užitečné a praktické.

### Příklad:

Objekt `Kvadr`, který je definován pomocí podstavy a výšky. Má metodu výpočtu objemu. Skript pomocí `alert` okénka zobrazí rozměry kvádru a objem.

*Výpis kvadr.html*

```
<SCRIPT LANGUAGE="JavaScript" SRC="obd2.js">
<!--

// -->
</SCRIPT>

<SCRIPT LANGUAGE="JavaScript">
<!--
function Kvadr(Podstava,mc) { /* objekt Kvadr */
  this.podstava=Podstava      // vlastnost podstava
  this.c=mc                   // vlastnost rozměr c, výška kvádru
  this.V=ObjemKvadru         // metoda V kvádru
}

function ObjemKvadru() {      // metoda kvádru počítá jeho V
  var v=0                     // objem
  v=this.podstava.S()*this.c   // v=this['podstava']['S']() * this['c']
  return(v)
}

var s=""
var obdelnik2 = new OOObdelnik(2,4) // instance objektu OOObdelnik
```

```

var kvadr = new Kvadr(obdelnik2,5) // instance kvadr objektu Kvadr
        /* alternativní zápis využívající polí */
s+="a="+kvadr.podstava.a          // kvadr['podstava']['a']
s+=" b="+kvadr.podstava.b          // kvadr['podstava']['b']
s+=" c="+kvadr.c                  // kvadr['c']

alert(s)
alert(kvadr.V())                 // zobrazí objem kvádru
// -->
</SCRIPT>
```

První značka SCRIPT zpřístupňuje kód JavaScriptu obsažený v souboru ob2.js. Chcete-li si jej znovu prohlédnout, nalistujte o několik stránek zpátky. Zjistíte, že je v něm deklarován objekt OOObdelnik, který má vlastnosti a, b a metody výpočtu obvodu a obsahu obdélníku. Podívejte se teď na to, co je uvedeno v rámci druhé značky SCRIPT. Funkce Kvadr deklaruje objekt Kvadr. Jeho vlastnosti jsou podstava, rozměr c a metodou je V. Všimněte si, že v této chvíli ještě není zjevné, že podstava bude objekt. Až z funkce ObjemKvadru, která je metodou objektu Kvadr, můžete mnohé poznat. Objem kvádru V totiž počítám jako obsah podstavy krát rozměr c. Při tvorbě konkrétní instance, bude podstavě přiřazena instance objektu OOObdelnik, čímž se stane proměnnou s tímto datovým typem. Ta zná své rozměry a umí vypočítat svůj povrch a obvod. Právě toho jsem využila ve funkci ObjemKvadru. Příkaz this.podstava.S() volá metodu objektu OOObdelnik počítající S. Přesně this znamená tento objekt, podstava jeho vlastnost podstava a S() její obsah. Jak by se totéž dalo zapsat s využitím asociativních polí je uvedeno v poznámce. Zbytek funkce ObjemKvadru by pro vás už měl být srozumitelný, proto se teď podívejme na samotný skript. Proměnná s je pomocná. Budu do ní vkládat rozměry kvádru. Dále jsem vytvořila instanci objektu OOObdelnik s názvem obdelnik2 a tu jsem hned použila při tvorbě instance Kvadr. Pamatujte si, že pokud chcete správně vytvořit instanci objektu, který obsahuje podobjekt, musíte vždy nejprve vytvořit tento podobjekt a ten pak použít jako argument při tvorbě objektu. Jiný postup nevede k vytyčenému cíli. Následuje zapsání rozměrů kvádru do proměnné s a zobrazení hodnoty s pomocí alert okénka. Pozor, rozměr a, resp. b vrací příkaz kvadr.podstava.a, resp. kvadr.podstava.b a ne kvadr.obdelnik2.a, resp. kvadr.obdelnik2.b! Myslet si, že právě druhá možnost je ta správná by byl hrubý omyl. Instance kvadr má vlastnost podstava a ne obdelnik2. Této vlastnosti byla pouze přiřazena při inicializaci hodnota proměnné obdelnik2. Dobře si to uvědomte, aby jste zbytečně nechybovali! V závěru skriptu ještě najeznete příkaz zobrazující alert okénko s objemem vytvořeného kvádru.

## Příkazy pracující s objekty

Mimo klíčového slova this, s kterým jste se setkali a seznámili v předchozích řádcích, existují ještě dva příkazy jež lze také použít pouze v souvislosti s objekty. Jsou to cyklus for...in a příkaz with. V Pascalu byste marně hledali příkaz for...in. U with byste byli naopak úspěšní, ale nechte se zmást stejným názvem příkazu v Pascalu a JavaScriptu. Oba sice umožňují zkrátit zápis, ale v Pascalu se with vztahuje k záznámům a v JavaScriptu k objektům. Až na cíl a název spolu tedy nemají nic společného.

### Příkaz for...in

```
for ( prom in NázevObjektu) {  
    příkazy for bloku  
}
```

Příkaz for...in je obdobou for cyklu pracující s objekty. Pro všechny vlastnosti a metody objektu, resp. instance s názvem NázevObjektu, vykonává příkazy bloku. Proměnná prom uchovává název vlastnosti či metody pro kterou je právě for...in blok vykonáván.

### Příklad:

Objekt Seznam rozšířený o metodu show, která vypíše všechny osoby včetně telefonu.

*Výpis seznam1.html (pouze samotný kód JavaScriptu)*

```
function Seznam() { /* objekt telefonní seznam */  
    this.Martina = "0501/494295" // název vlastnosti je jméno  
    this.Lenka = "02/334578" // hodnota telefon  
    this.Lee = "0501/354266"  
    this.show = Show  
}  
  
function Show() { /* zobrazí všechny vlastnosti včetně hodnot  
    uvedené před metodou show */  
    for ( var iter in this){ // pro vlastnosti a metody tohoto objektu  
        if (iter=="show") break // jestliže je iter show opust' cyklus  
        document.write("<BR>" +iter+ " " +this[iter]) // vypíš  
    }  
}  
  
var seznam = new Seznam() // vytvoření instance objektu Seznam  
seznam.show()
```

Skript se skládá z funkce Seznam() deklarující objekt, funkce Show(), tvorby instance seznam a volání metody show, která zobrazuje náš telefonní mini seznam. Metoda show je v deklaraci objektu spojena s funkcí Show(). Show() pracuje následovně. Pro všechny vlastnosti a metody objektu, který jí využívá, vykonává příkazy for bloku. Jestliže má iter, obsahující název právě aktuální vlastnosti, hodnotu "show" cyklus je „násilně“ ukončen pomocí break. Další příkaz vypisuje název vlastnosti, resp. metody iter a její hodnotu do zobrazeného dokumentu. V tomto případě je iter deklarována a následně je jí přiřazen řetězec "Martina" (tj. název první vlastnosti objektu). If podmínka není splněna a tak je vykonán další příkaz cyklu document.write. Ten vypíše do zobrazeného dokumentu svůj argument. HTML značky v něm mají stejný význam jako HTML značky. <BR> tedy odřádkuje a poté se zobrazí hodnota proměnné iter (řetězec Martina) mezera hodnota this[iter] (řetězec 0501/494295 tj. hodnota vlastnosti Martina). Při dalším průchodu for cyklem je proměnné iter přiřazen název následující vlastnosti objektu. If podmínka není splněna (iter je "Lenka") a tak je vykonán další příkaz cyklu. Ten odřádkuje a poté zobrazí hodnotu proměnné iter (řetězec Lenka) mezeru a hodnotu this[iter] (řetězec 02/334578). Při dalším průchodu for cyklem je proměnné iter přiřazen název následující vlastnosti objektu. If podmínka není splněna (iter je "Lee") a tak je vykonán další příkaz cyklu. Ten odřádkuje a poté zobrazí hodnotu proměnné iter (řetězec Lee) mezeru a hodnotu this[iter] (řetězec 0501/354266). Při dalším průchodu for cyklem je proměnné iter přiřazen název následující vlastnosti respektive metody objektu. If podmínka je splněna (iter je "show") a tak je cyklus násilně přerušen. Dá se říci, že metoda show

vypisuje názvy všech vlastností a metod objektu včetně hodnot, které jí předchází. Funkce Show() je natolik obecná, že ji můžete použít při definici metody show v jakémkoliv objektu. Pokud budete chtít zobrazit pouze všechny vlastnosti, stačí ji umístit za ně jako první metodu.

### Příkaz with

```
with (NázevObjektu) {  
    příkazy  
}
```

NázevObjektu je název objektu nebo instance. Příkaz with umožňuje zkrácený zápis přístupu k jednotlivým položkám objektu. Odkazy na vlastnosti či metody objektu uvnitř with bloku sestávají pouze z jejich názvů. Např. nepíšeme NázevObjektu.v1, ale pouze v1, což je nejen kratší, ale mnohdy i přehlednější.

### Příklad:

Jak lze vhodně změnit skript v souboru kvadr.html s využitím příkazu with?

Výpis souboru kvadr.html naleznete ve stati vysvětlující vnořené objekty. Použití příkazu with se v něm nabízí hned na dvou místech. Kód

```
function ObjemKvadru() {          // metoda kvádru počítá jeho V  
    var v=0                         // objem  
  
    v=this.podstava.S()*this.c      // v=this['podstava']['S']() * this['c']  
    return(v)  
}
```

můžeme nahradit kódem

```
function ObjemKvadru() {          // metoda kvádru počítá jeho V  
    var v=0                         // objem  
  
    with (this) {                  // s tímto objektem - this  
        v=podstava.S()*c           // píšeme už jen podstava.S(), c  
                                // a ne this.podstava.S(), this.c  
    }  
    return(v)  
}
```

A místo

```
s+="a="+kvadr.podstava.a          // kvadr['podstava']['a']  
s+=" b="+kvadr.podstava.b         // kvadr['podstava']['b']  
s+=" c="+kvadr.c                 // kvadr['c']  
  
alert(s)  
alert(kvadr.V())                  // zobrazí objem kvádru
```

je určitě přehlednější psát

```
with (kvadr) {                    // s objektem kvadr  
    s+="a="+podstava.a            // píšeme jen podstava.a a ne kvadr.podstava.a  
    s+=" b="+podstava.b           // jen podstava.b a ne kvadr.podstava.b  
    s+=" c="+c                   // jen c a ne kvadr.c  
    alert(s)  
    alert(V())                  // píšeme jen V() a ne kvadr.V()  
}
```

Tímto způsobem upravený skript kvadr.html jsem uložila do souboru wkvadr.html.

### Cvičení:

- 1) Vytvořte následující objekty a poté od každého alespoň jednu instanci. Vypište u prvních dvou hodnoty jejich vlastnosti a výsledek volání metody. U instance posledního objektu Hranol vypište obsah podstavy S a objem V. (*objekty.js, kobjekty.html*)
  - a) Lichobezník s vlastnostmi a, c, v a metodou výpočtu obsahu S, kde a a c jsou délky rovnoběžných stran a v je jejich vzdálenost, tj. výška lichoběžníku.
  - b) Rovnobězník s vlastnostmi a, v a metodou výpočtu obsahu S, kde a je délka strany a v je výška rovnoběžníku k ní kolmá.
  - c) Hranol s vlastnostmi podstava (která může být jak lichoběžník tak rovnoběžník) a v a metodou výpočtu objemu V, kde v je výška hranolu.
- 2) Vytvořte objekt umožňující pracovat s aritmetickou posloupností, která je dána prvním členem posloupnosti a a diferencí d. Konkrétně musí mít metodu na výpočet n-tého člena posloupnosti a na výpočet součtu prvních n členů. Hodnota n je u obou metod parametr. Další metody tohoto objektu nechávám na vašem uvážení (např. zda je zadané číslo členem posloupnosti a kolikátým atd.). (*posl.js*)
- 3) Vytvořte skript pracující s objektem z předchozího příkladu. Po kliknutí na tlačítko s nápisem n načte celé, kladné, nenulové n a pomocí alert okénka zobrazí n-tý člen i součet prvních n členů posloupnosti. Po stisknutí tlačítka myši nad buttonem s nápisem a resp. d, zobrazí hodnotu a resp. d, při uvolnění tlačítka myši nad buttonem (tj. stisknu tlačítko myši, nejedu nad element a pustím ho) dovolí změnit hodnotu a resp. d. (*kposl.html*)
- 4) Vytvořte objekt (asociativní pole) Slovnik, který bude obsahovat anglická slovíčka s jejich českými ekvivalenty. (*slova.js*)
- 5) Objekt Slovnik rozšiřte o následující metody: (*slovnik.js*)
  - a) show – zobrazí všechna slovíčka
  - b) CtoE(slovo) – pokud je slovo ve slovníku, vrací jeho anglický ekvivalent, jinak vrací řetězec "Slovo nenalezeno".
  - c) EtoC(slovo) – pokud je slovo ve slovníku, vrací jeho český ekvivalent, jinak vrací řetězec "Slovo nenalezeno".

Ověřte si funkčnost těchto metod ve skriptu. (*slovnik.html*)

## Objekty JavaScriptu

V JavaScriptu, narozdíl od Pascalu, existují objekty, které vytvořili jeho autoři. Jediný úkol, který nám přenechali, je naučit se s nimi pracovat. Jelikož jsou automaticky obsaženy v prostředí JavaScriptu, lze je nazývat objekty JavaScriptu. Když se pořádně zamyslíte, možná přijdete na to, že už jste se zaručeně s jedním objektem JavaScriptu setkali. No ano, stačí si třeba vzpomenout na to, jak jste měnili barvu pozadí příkazem `document.bgColor = color`. Už vidíte, že jím byl `document`. Tato kapitola vám umožní se setkat se všemi objekty JavaScriptu. Jejím cílem není vás postupně zahrnout výčtem vlastností a metod všech objektů, ale udělat si o nich ucelenou představu. Pochopit základní smysl a s tím spojenou možnost využití toho kterého objektu. Opět dám přednost snaze vám něco vysvětlit a ukázat na příkladech, před suchou teorií. Jelikož však bez teorie není praxe a není nazbyt mít někdy po ruce relativně přesný popis objektů JavaScriptu, můžete jej krom jiného najít v práci Petra Kočičky. Většina z vás ji už určitě objevila při prozkoumávání přiloženého média. Stačí pomocí prohlížeče zobrazit soubor `index.html` v adresáři `KOCICKA`. Pořádně se v ní rozkoukejte, protože předpokládám, že při řešení cvičení si vyhledáte přesný popis potřebných funkcí právě zde (pokud nemáte k dispozici nějakou jinou aktuálnější k tomu vhodnou literaturu). Ted' se konečně vraťme k našemu hlavnímu tématu.

Objekty JavaScriptu tvoří tři základní skupiny a to objekty HTML, objekty prohlížeče a vestavěné objekty. Vestavěné objekty nemají nic společného s web stránkami či HTML jazykem. Představují rozšíření JavaScriptu o pole, základní matematické funkce a konstanty, práci s datumem či řetězci. Objekty prohlížeče jsou oproti tomu těsně spojeny s prohlížečem a objekty HTML zase s HTML prvky stránky a HTML jazykem. Seznámení se s nimi vás čeká na několika následujících stránkách.

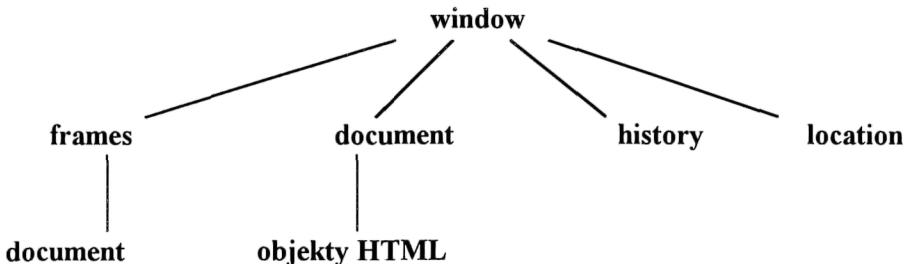
Ještě než začnu samotný výklad, chtěla bych zdůraznit, že objekty JavaScriptu jsou objekty. Lze s nimi tedy pracovat s užitím příkazů `with` a `for...in`, resp. klíčového slova `this`. Na jejich vlastnosti a metody se můžeme odkazovat stejně tak, jak bylo uvedeno v předchozí kapitole, konkrétně tečkovou notací a zápisem postaveném na faktu, že objekty jsou asociativní pole. A pokud vám doposud není zcela jasné k čemu jsou vlastně objekty dobré a chybí vám vůbec ucelenější představa na toto téma, nezoufejte. Po nastudování objektů JavaScriptů se určitě ucelí.

## Objekty prohlížeče

Objekty prohlížeče odráží prostředí prohlížeče. Představují souhrnné informace a akce, které jsou nutně spojené s určitou Web stránkou. Každá stránka je zobrazena v okně (`window`) prohlížeče, které má svou historii (`history`) a zná adresu (`location`) aktuálního dokumentu. Okno prohlížeče musí mít k dispozici také popis zobrazeného dokumentu (`document`). Dokument se pak skládá s nějakých HTML prvků. Ty mají přímou spojitost s HTML objekty. A když vám ted' řeknu, že anglická slova uvedená v závorkách jsou samotné názvy objektů prohlížeče, nebude pro vás pro-

blém představit si hierarchii zmíněných objektů prohlížeče. Taky tvrzení, že HTML objekty jsou podobjekty některých objektů prohlížeče, by vás nemělo udivovat. Ostatně dalo se předpokládat.

Poslední objekt prohlížeče, o kterém ještě nepadlo ani slovo, je frames. Při použití framů na HTML stránce umožňuje odkázání na jednotlivé rámy. Je podobjektem window stejně jako objekty document, history a location. Takže window je na vrcholu hierarchie objektů prohlížeče, kterou lze schematicky zobrazit asi takto:



### Objekt window

Každé okno, které je právě otevřené, má odpovídající objekt window. Všechny ostatní objekty jsou jeho následníky. JavaScript udržuje myšlenku aktuálního okna, takže téměř žádné odkazy na podobjekty aktuálního okna se na něj nemusí odkazovat explicitně. Díky tomu jste při změně barvy pozadí psali document.bgColor a ne window.document.bgColor. Lze tedy vypouštět slovo window a místo window.vlastnost, resp. window.metoda(parametry), psát jen vlastnost, resp. metoda(parametry).

Mezi vlastnosti objektu window patří krom jeho podobjektů např. i status, která obsahuje stávající hlášení ve stavovém řádku okna prohlížeče.

Metody jsou např. vám důvěrně známé alert(string), prompt(string, input), ale i confirm(string), zobrazující podobné okno jako prompt, pouze bez textového pole. Dále close() zavírá okno prohlížeče a naopak open(URL) je otvírá se stránkou na URL. Open má ještě dva nepovinné parametry. První je název tvořeného okna a druhý nám umožňuje říci, zda v něm zobrazovat panely nástrojů atd. Užitečná je metoda setTimeout(exp, n), která vždy po n milisekundách vyhodnotí výraz exp. Často je exp volání funkce. I když se nejedná zdaleka o celkový výčet metod, je z něj vidět, že objekt window má opravdu celkem dobrý vztah s prohlížečem.

### Objekt document

Každé okno má odpovídající objekt document. Po window je to nejpoužívanější a nejdůležitější objekt prohlížeče.

Jeho vlastnosti jsou spojeny se samotným dokumentem. Část z nich jsou objekty HTML, o kterých pojednává celá další část, zbytek obsahuje atributy HTML značek kostry daného dokumentu. Např. BODY má atribut bgColor, document má vlastnost bgColor stejného významu atd. Vlastnost title vrací text uzavřený ve stejnojmenných značkách, URL adresu dokumentu, ...

Z metod už jste používali write(string). Velmi často je před ní volána metoda open() připravující dokument na příson nových dat a za ní close(), jež je opakem open(). Write by se správně měla vždy umisťovat mezi open a close, ale sami jste se přesvědčili, že v některých případech to z hlediska správné funkčnosti není nutné.

## Objekt frames

Pomocí framů je možné rozdělit okno prohlížeče na několik částí a v každé z nich zobrazit jinou HTML stránku. Prakticky se to dělá tak, že si zpravidla nejprve vytvoříte HTML stránky str0.html až strn.html a teprve pak zastřešující stránku frame.html v níž pomocí HTML značek definujete počet framů a jejich obsah. Stránky str1.html až strn.html jsou na sobě zcela nezávislé. Objekt frames má přímou spojitost s framy dokumentu a podstatně rozšiřuje jejich využití. Stránky jednotlivých framů spolu mohou s jeho využitím nejen sousedit, ale i spolupracovat. Jak toho lze docílit? V každém framu je zobrazen dokument, který má svůj odpovídající objekt document. Objekt document je tedy podobjekt objektu frames. Objekt frames má všechny své frame očíslované. Zdálo by se, že frames[0].document.vlastnost vrátí požadovanou vlastnost dokumentu zobrazeného v nultém framu, jenže tak tomu není i když jsme na správné stopě. Pokud chceme, aby dokument str0.html zobrazený v nultém framu mohl získat hodnotu vlastnosti dokumentu str1.html zobrazeného v prvním framu musíme přímo v těle dokumentu str0.html použít příkaz parent.frames[1].document.vlastnost popřípadě top.frames[1].document.vlastnost. Slovo parent se dá přeložit jako rodič a společně s top se jedná o alternativní název pro rodičovský dokument frame.html obsahující definici framů. Je tedy nutné se nejprve pomocí parent či top odkázat na stránku s definicí framů a až pak na příslušný frame. Samotný frame lze také jednoznačně identifikovat jménem uvedeným v atributu NAME a v příkazech frames[1] uvádět právě toto jméno. O atributu NAME se více dočtete o několik stran dále.

Shrňme si ted' slova předchozího odstavce. HTML stránky zobrazené ve dvou různých framech spolu mohou spolupracovat. Dokonce jsou často navrženy právě k tomuto účelu, to když je například v jednom framu zobrazeno menu s hypertextovými odkazy a v druhém se zobrazuje vybraná stránka. Pokud může HTML stránka zobrazená v jednom framu získávat a měnit svoje vlastnosti pomocí příkazu document.vlastnost, může měnit a získávat vlastnosti HTML stránek zobrazených v ostatních framech pomocí parent.frames[n].document.vlastnost, kde n je číslo framu. Velmi často se frames[n] v předchozím příkazu nahrazuje názvem framu a slovo parent slovem top, které má podobný význam jako parent.

## Ostatní objekty prohlížeče

Všechny ostatní objekty prohlížeče jsou, zrovna tak jako document a frames, podobjekty window. Jedná se konkrétně o history a location.

History se používá při odkazu na seznam už navštívených stránek. Má pouze jedinou vlastnost a to length, která udává počet URL právě obsažených v seznamu historie prohlížeče. Mezi jeho metody patří go(kam), která umožňuje navigaci seznamem historie. Argument kam může být celé

číslo. Např. go(1) má stejný účinek jako kliknutí na tlačítko Forward (Vpřed). Je-li kam kladné, jedná se o pohyb historií vpřed, je-li záporné, vzad.

Objekt location popisuje URL dokumentu. Příkaz location i location.href vrací (stejně tak jako document.URL) úplnou adresu zobrazeného dokumentu. Vlastnosti reprezentují jednotlivé, běžně rozlišované, části URL. Např. protocol vrací použitý protokol, pathname cestu, ...

### **Příklad:**

Při načtení souboru win.html se otevře nové okno prohlížeče bez ovládacích panelů s nápisem (jeho text je nepodstatný) a tlačítkem close. Pokud na tlačítko kliknete, bude okno zavřeno.

*Výpis win.html (pouze samotný kód JavaScriptu)*

```
var okno=null      // nové okno prohlížeče
var s=""           // obsah nového okna
                  // do s dávám řetězec popisující HTML stránku
s+="<HEAD><TITLE>Ahoj</TITLE></HEAD>"
s+=""
s+="

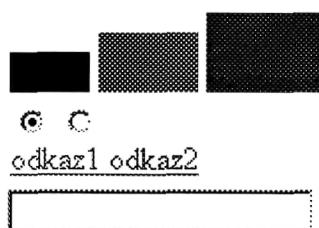
## Ahoj</H2> <BR>" s+="" s+="" s+="</FORM>" s+="</BODY></HTML>" okno=open("", "", "toolbar=no, directories=no, menubar=no, height=200, width=200") // v kódu jeden řádek //prázdné okno bez názvu, nezobrazuj toolbar // directories, menubar, výška a šířka 200 okno.document.open() // metoda open "otevírá document pro zápis" okno.document.write(s) // zápis okno.document.close() // metoda close documentu "konec zápisu"


```

Všimněte si, že pomocí metody write lze zobrazovat řetězec obsahující jakýkoliv HTML kód včetně popisu celého dokumentu. U HTML dokumentů je tedy možné smazat veškeré značky v těle a zobrazovat je JavaScriptem. Dále bych vás chtěla upozornit na proměnnou okno. Bylo ji nutné vytvořit. Bez její pomoci by sice nebyl problém nové okno prohlížeče otevřít, ale přišli bychom o možnost s tímto oknem jakkoliv pracovat. Poslední zajímavostí tohoto skriptu je vlastní příkaz open. Sami si zkuste např. umazat z něj toolbar=no či no nahradit yes. Co se stalo?

## **Objekty HTML**

Objekty HTML jsou podobjekty objektu document a odráží se v nich nejen HTML značky dokumentu, ale i jejich struktura. Každý HTML prvek má odpovídající objekt HTML. Pro snadnější pochopení objektů HTML jsem vytvořila konkrétní stránku zobrazenou na obrázku.



Prohlédněte si její zdrojový kód.

*Výpis html.html(pouze tělo dokumentu)*

```
<BODY>
<IMG name="obr0" src='OBRAZKY/black.jpg' width='40' height='20'>
<IMG name="obr1" src='OBRAZKY/red.jpg' width='50' height='30'>
<IMG name="obr2" src='OBRAZKY/blue.jpg' width='60' height='40'>
<BR>
<FORM name="moznosti">
<INPUT name="volba" type=radio value="0" checked>
<INPUT name="volba" type=radio value="1">
</FORM>
<A HREF='sablona.html'> odkaz1 </A>
<A HREF='seznam1.html'> odkaz2 </A>
<FORM name="pole">
<INPUT name="info" type=text>
</FORM>
</BODY>
```

Jedná se o běžnou HTML stránku obsahující nějaké tři obrázky, formulář radio, dva odkazy a nakonec ještě jeden formulář s textovým polem. JavaScript má, jak už víte, pro všechny HTML prvky své objekty HTML.

Obrázky reprezentuje objekt images, který tvoří pole. Např. document.images[0].width vrací hodnotu vlastnosti width prvního obrázku, document.images[2].src zase umístění třetího posledního obrázku. Takže index v závorce u images udává pořadové číslo obrázku a za tečkou následuje název atributu, který nás zajímá. Dávejte si pozor na fakt, že JavaScript indexuje první prvek pole vždy číslem nula.

Formuláře reprezentuje objekt forms, který tvoří pole. V rámci formuláře umožňuje jednotlivé jeho položky rozlišit pole elements. Např. document.forms[0].elements[0].value vrací hodnotu value prvního elementu v prvním formuláři, což je 0. Pokud by nás zajímalo zda je zaškrtnuté druhé radio tlačítko, napíšeme document.forms[0].elements[1].checked. Je-li vybráno, obdržíme odpověď true. Hodnota této vlastnosti odráží aktuální stav formuláře na zobrazené stránce. Veškeré objekty HTML jsou aktualizovány zároveň se zobrazeným dokumentem. Právě to nám umožňuje zjistit, které radio tlačítko má uživatel právě vybráno, co napsal do textového pole formuláře atd.

Hypertextové odkazy reprezentuje objekt links, který tvoří také pole. Pracuje se s ním očekávaným způsobem. Například document.links[0].href je hodnota atributu href prvního odkazu v dokumentu.

Ukotvení reprezentuje objekt anchors, který tvoří pole.

Všechny uvedené objekty mají vlastnost length. Například document.images.length je 3, protože na stránce je právě tolik obrázků, document.anchors.length je 0, což značí, že dokument neobsahuje ani jedno ukotvení.

Právě teď jste se seznámili s nejdůležitějšími objekty HTML. Víte, že obsahují ty nejaktuльнější informace vztahující se k určitému dokumentu. Objekty HTML mohou tyto informace nejen vhodně využívat, ale v některých případech i měnit. Určitě jste se s tím setkali. Třeba takové

jdoucí hodiny v textovém poli vyžadují aktualizovat hodnotu value tohoto pole po určitých časových intervalech. Kdyby nemohl JavaScript přiřazovat do atributu value, nikdy by se s pomocí JavaScriptu na HTML stránkách žádné hodiny nerozešly.

### Příklad:

Ukázka změny atributu href odkazu po kliknutí na tlačítko.

*Výpis odkaz.html(pouze tělo dokumentu)*

```
<BODY>
<A HREF='sablona.html'> odkaz </A>
<FORM>
<INPUT type=button value='Klikni'
       onClick='document.links[0].href="bgfocus.html"'>
</FORM>
</BODY>
```

Při otevření stránky odkaz.html odkaz směřuje na soubor sablona.html umístěný v též adresáři jako odkaz.html. Jakmile uživatel klikne na tlačítko provede se příkaz měnící hodnotu atributu href, takže pomocí odkazu nyní otevřete dokument bgfocus.html. V praxi asi využijete možnost měnit cíl odkazů spíše na základě zatržení radio volby či podle hodnoty výběrového pole SELECT než po kliknutí na tlačítko. Účelem tohoto příkladu je vás přesvědčit na vlastní oči, že JavaScript opravdu může měnit hodnoty některých atributů HTML značek.

### Příklad:

Ukázka „animace“ obrázků.

*Výpis anim.html(pouze samotný JavaScript)*

```
var pom=true           // určuje který obrázek zobrazovat
                  /* každý používaný obrázek je nutné deklarovat */

obr0=new Image(65,100)      // vytvoření nového obrázku
obr0.src='OBRAZKY/0.gif'    // naplnění atributu src
obr1=new Image(65,100)
obr1.src='OBRAZKY/1.gif'

function Init () { /* zobrazení první stránky */
var s='';
s+='BR';
s+='CENTER';
s+='IMG SRC="OBRAZKY/Transpar.gif" WIDTH="65" HEIGHT="100"';
s+='</CENTER>';
document.write(s);
}

function Rotate () {      /* samotná "animace" */
if (pom)                // když je pom true zobraz obr1
  document.images[0].src = obr1.src
else                    // jinak zobraz obr2
  document.images[0].src = obr0.src
pom=!pom                // zneguj pom
setTimeout("Rotate()",1000) // za 1000 milisekund spust Rotate()
}

Init ()                  // inicializuj
Rotate ()                // rotuj
```

Skript strídavě zobrazuje obrázek s číslem nula obr0 a jedna obr1. Oba jsou instancemi objektu Image. Např. `obr0 = new Image(65,100)` vytváří proměnnou obr0, která bude obsahovat nějaký obrázek o rozměrech 65x100 bodů. Slovo nějaký jsem použila zcela záměrně, protože až příkaz `obr0.src='OBRAZKY/0.gif'` dodává vlastnosti src adresu tohoto obrázku, čímž ho konkretizuje. Deklaraci obr1 lze vysvětlit podobným způsobem. Po provedení deklaračních příkazů je spuštěna funkce Init(), zobrazující stránku s „průhledným“ obrázkem. Následně je vykonáván kód funkce Rotate(). Proměnná pom je true a proto je zobrazena jednička. Všimněte si, že ve skutečnosti byla změněna hodnota atributu src a to způsobilo překreslení obrázku. Příkaz `pom!=!pom` přiřazuje pom negaci sebe sama, takže pom je false. Za 1000 milisekund je opět volána funkce Rotate(). Proměnná pom je false a proto je zobrazena 0, ... Uvědomte si, že pom musí být globální. Kdyby byla lokální, popisovaný skript by zobrazoval stále stejné číslo. Sami si poslední tvrzení zdůvodňete.

### Atribut NAME

Dříve než na stránky HTML pronikl JavaScript, nemělo používání atributu NAME příliš smysl, takže mé značky zůstávaly většinou bezjmenné. Až JavaScript mě přiměl k jejich pečlivému pojmenovávání. Proč?

Jednou se mi stalo, že jsem si vytvořila stránku obsahující (pro tuto chvíli nedůležitý) text, pod nímž byl formulář s textovým polem a tlačítkem. Při kliknutí na tlačítko jsem zjistila obsah textového pole a vyhodnotila jej. Jelikož se jednalo o první formulář v dokumentu, hodnotu textového pole obsahoval `document.forms[0].elements[0].value`. Jenže jsem byla donucena hned na začátek stránky přidat další formulář. Naštěstí mi došlo, že aby dokument pracoval tak jak má, musím na všech místech výskytu ve skriptu `document.forms[0].elements[0].value` nahradit kódem `document.forms[1].elements[0].value`. Po nějaké době jsem na začátek stránky přidala další formulář. Až když se můj dokument choval podivně, začala jsem v něm hledat chybu. Přišla jsem na to, že nevyhodnocuji po kliknutí na tlačítko textové pole které chci, ale to předchozí. Řetězec `forms[1]` jsem přepsala na `forms[2]` a skript začal správně fungovat. Problém byl samozřejmě v tom, že přidáním nového formuláře na začátek stránky se změnil index formuláře s nímž jsem spolupracovala. Takže jen malá změna HTML souboru, může vynucovat úpravy kódu JavaScriptu. Tehdy jsem začala být přinejmenším rozhořčená a napadla mě jedna otázka. Musím se opravdu při používání objektů HTML spoléhat na něco tak pomíjivého jako je pořadové číslo toho kterého objektu obsaženého na stránce? Odpověď na položenou otázku zní ne. Stačí začít pojmenovávat HTML značky.

Atribut NAME je vám patrně známý a právě on je klíč k řešení nastíněného problému. JavaScript totiž umožňuje přistupovat k jednotlivým atributům HTML značek pomocí jejich jména. V souboru `html.html`, o kterém byla řeč hned v prvních řádcích této statě, jsem značky vhodně pojmenovala. První obrázek má název `obr0`. Hodnotu jeho atributu `width` vrací jak `document.images[0].width` tak `document.obr0.width`. Obdobně obsah textového pole lze získat pomocí `document.forms[1].elements[0].value`, protože je to druhý formulář v dokumentu a v jeho rámci první element, nebo pomocí `document.pole.info.value`, protože formulář má název pole a textové

políčko v něm info. Jak sami vidíte, používat atribut NAME není nikterak obtížné, máte-li na paměti strukturu objektů HTML. Jen si musíte dávat pozor na jedinečnost názvů v rámci jednoho dokumentu. Sami určitě intuitivně cítíte, že např. stejné jméno obrázku a formuláře či dokonce dvou formulářů, by mohlo způsobit problémy. Jak by JavaScript poznal, který HTML element máte zrovna na mysli?

### Použití klíčového slova this

Klíčové slovo this označuje tento objekt. Při tvorbě vašich vlastních objektů se vám stalo určitě pomocníkem. Jak vám může být nápomocno ve spojení s objekty HTML, ukazuje jednoduchý ilustrační příklad.

#### Příklad:

Po stisknutí tlačítka se v textovém poli objeví jeho popis.

*Výpis this.html(pouze tělo dokumentu)*

```
<BODY>
<FORM name='f'>
<INPUT name='pole' type=text >
<INPUT name='tlacitko' type=button value='123'
       onClick='document.f.pole.value=this.value' >
</FORM>
</BODY>
```



V dokumentu je definován formulář s textovým polem a tlačítkem. Po stisknutí tlačítka je vykonán příkaz document.f.pole.value=this.value. Slovo this zde znamená tento objekt HTML, takže this.value je hodnota atributu value značky v níž se kód nachází. Bez použití this bych musela místo this.value psát document.f.tlacitko.value nebo document.forms[0].elements[1].value.

Obecně se this zpravidla vyskytuje v kódu JavaScriptu ošetřujícím událost uvedeným přímo v HTML značce a značí příslušný HTML objekt (podobjekt) k ní neodmyslitelně patřící.

### Metody HTML a události

Objekty HTML mají metody, které úzce souvisí s událostmi. Lze-li v rámci HTML značky použít událost onNazev, pak objekt HTML spojený s HTML značkou má metodu nazev(). Konkrétně tvoří „dvojice“ onFocus a focus(), onSelect a select(), ... Jestliže onSelect je vyvolána po označení textu ve vstupním poli, tak select() způsobí označení textu ve vstupním poli. Jestliže onFocus je vyvolána při aktivaci elementu, tak focus() element aktivuje. Stejná vazba je i mezi dalšími událostmi a příslušnými metodami. Jak vidíte, tyto metody jsou schopny vykonat akci vyvolávající událost jako by to udělal sám uživatel.

#### Příklad:

Dokument znemožňuje změnit hodnotu textového pole, jež lze aktivovat buď běžným způsobem nebo kliknutím na tlačítko.

```
Výpis metody.html(pouze tělo dokumentu)
<BODY>
<FORM name='f'>
<INPUT name='pole' type=text value='123'
       onFocus='this.blur(); alert("Nepiš sem.")'>
<INPUT name='tlacitko' type=button value='Klik'
       onClick='document.f.pole.focus()'>
</FORM>
</BODY>
```

Při vstupu do textového pole je vykonán kód události onFocus. Příkaz this.blur() způsobí opuštění textového pole a alert("Nepiš sem.") zobrazí alert okénko. Kliknutí na tlačítko je také pokus o vstup do textového pole, proto spustí opět kód události onFocus. Opravdu je tedy metoda focus() (i ostatní metody tohoto druhu) ekvivalentní akci uživatele a dokáže vyvolat událost s ní spojenou, pokud existuje. Jestliže bychom neměli k dispozici metodu focus() a blur(), bylo by zadání JavaScriptem neřešitelné.

## Vestavěné objekty

Vestavěné objekty nemají nic společného s HTML jazykem, prostředím aktuálního prohlížeče nebo ničím viditelným. Neexistuje mezi nimi a objekty HTML nebo prohlížeče žádný vztah. Co do počtu jsou čtyři a to Array (pole), String (řetězec), Date a Math. Pole nebo také Array je částečně obdobou tohoto datového typu běžných programovacích jazyků. Objekt String sdružuje zejména metody pracující s řetězci. Date přináší mnohem víc než aktuální datum. Math nabízí mimo jiné běžné matematické funkce a konstanty. Tato stať je tematicky rozdělena na čtyři hlavní části. Každá stručně popisuje jeden vestavěný objekt JavaScriptu.

### Array (pole)

Pojem pole je vám určitě důvěrně známý z programovacího jazyka s kterým jste se setkali. I v této knize slovo pole několikrát padlo v předchozí kapitole pojednávající o tvorbě objektů. Od tut víte, že každé pole je objekt a zároveň každý objekt je pole. Na tomto místě získáte hlubší znalosti o polích. Naučíte se je tvorit, pracovat s nimi, ..., což značně rozšíří vaše „programátor-ské“ možnosti v JavaScriptu. Budete moci bez problémů načítat větší množství dat a následně s nimi pohodlně pracovat. Ale ted' se soustřeďte na samotný výklad.

Každé pole se skládá z indexovaných prvků. Indexy mohou být bud' čísla nebo u asociativních polí obecné řetězce. Při tvorbě nového pole je nutné použít slova new. Konkrétně vypadá takto:

```
var NazevPole = new Array() // var u deklarace polí velmi často chybí
```

Pole má název NazevPole a je jednorozměrné. Do závorky za názvem objektu Array lze uvést číslo představující předpokládanou délku pole, ale není to nutné. JavaScript se totiž na pole dívá jako na dynamický objekt, který je možné kdykoliv podle potřeby rozširovat. Kdyby bylo pole statické, v deklaraci by nesměla chybět jeho délka, která by byla závazná.

V této chvíli vás určitě zajímá, jak přiřadit konkrétní hodnotu jednotlivým prvkům pole. Pomocí běžného přiřazovacího příkazu NazevPole[index]=hodnota. První index pole je u JavaScriptu 0,

pokud to není pole asociativní. Asociativní pole se od „běžných“ liší pouze použitými indexy. Vytvořme si nyní, pro větší názornost, od každého pole jedno.

„běžné“ pole	asociativní pole
<pre>var a=new Array() a[0]=12 a[1]=5 a[2]= "Lenka" /* pole se stejným obsahem lze vytvořit také var a=new Array(12,5, "Lenka") */</pre>	<pre>var asoc=new Array() asoc["ČR"]="Praha" asoc["Slovensko"]="Bratislava" asoc["lidi"]=123456789 /* neexistuje alternativní možnost pro vytvoření asociativního pole o stejných indexech a obsahu */</pre>

Všimněte si, že hodnoty prvků pole nemusí být stejného datového typu, i když tomu tak v praxi zpravidla je. Například v Pascalu by se vám předchozí pole nepovedla vůbec vytvořit, protože v tomto programovacím jazyce musí být všechny prvky pole stejného datového typu.

Jelikož Array je objekt, logicky z toho plyne, že má alespoň jednu vlastnost nebo metodu. Z programovacího jazyku, s kterým jste se setkali, nejste určitě na nic takového zvyklí a možná vám to přijde nepřirozené. Pokud ano, uvědomte si, že objekt je kolekce vlastností a metod a Array je objekt. Větu si několikrát zopakujte, aby se vám vstípila do paměti a pak čtěte dál. Ještě bych vás chtěla upozornit, že vlastnosti a metody Array fungují očekávaným způsobem pouze pro „běžná“ pole. U asociativních polí vrací často nulu nebo prázdný řetězec.

Objekt Array má vlastnost length udávající počet prvků pole. Např. a.length je 3, protože pole a má právě tři prvky.

Z metod připomenu za všechny join(), která vrací řetězec prvků pole. Jednotlivé prvky jsou navzájem odděleny čárkou. Např. a.join() vrací "12,5,Lenka".

Asi jste si všimli, že veškerá má slova se doposud týkala jednorozměrných polí. Jednorozměrné nebo-li jednodimensionální pole nám JavaScript přímo nabízí. Ale co vícerozměrná pole? Do dvourozměrných polí se běžně načítají matice. Nabízí nám JavaScript aspoň ty? Bohužel ne.

Jenže co to vlastně dvourozměrné pole je? Úplné pochopení odpovědi na otázku je velmi důležité. Na základě ní vás už nebude pálit, že JavaScript dvourozměrná pole nezná, vytvoříte si je prostě sami. V následujících rádcích, i když to tak zpočátku nevypadá, naleznete odpověď na tuto otázku a praktickou ukázkou konstrukce dvourozměrného pole.

Pojem tabulka každý z vás určitě zná. Obecně může mít tabulka nejen různé rozměry, ale dokonce nemusí mít ani obdélníkový rozměr. Zaměřme se teď speciálně na takové tabulky, které mají v každém řádku stejný počet datových buněk (prvků) a nadále uvažujme pouze je. Konkrétní tabulka může vypadat třeba takto:

a	b	c	d
e	f	g	h

Nazveme si ji třeba T. Tabulka T má 2 řádky a 4 sloupce. Řádky a sloupce můžeme očíslovat například bud' od nuly nebo od jedničky. Domluvme se na první variantě. Jaká je v tuto chvíli odpověď na dotaz, kde se v tabulce T nachází h? Hodnota h se v tabulce T nachází v prvním

řádku a třetím sloupcí. Odpověď lze i zkráceným zápisem  $T[1][3]='h'$ , který pomocí několika znaků říká to, co předchozí věta. Všimněte si, že je nutné v odpovědi udávat jak index řádku, tak index sloupce. Jen kdyby tabulka obsahovala právě jeden řádek, stačilo by uvést pouze index sloupce. No dobré, ale co má společné tabulka a pole? Možná víc, než si myslíte.

Jednorozměrné n-prvkové pole si lze názorně představit jako jednořádkovou tabulkou o n sloupcích v jejichž buňkách jsou hodnoty příslušných prvků. Například pole a, vytvořené jako ukázka „běžného“ pole, má v nultém sloupci 12, protože  $a[0]=12$ , v prvním sloupci 5, protože  $a[1]=5$  a v druhém sloupci Lenka, protože  $a[2]= "Lenka"$ . U dvourozměrných polí mají, na rozdíl od jednorozměrných, všechny prvky dva indexy (u třírozměrných tří, ...) stejně jako u víceřádkových tabulek. Předchozí tabulka a vůbec všechny tabulky se dají chápat jako dvourozměrná pole. Každý řádek tabulky je vlastně jednorozměrné pole. Pomocí pole v němž nultý prvek obsahuje pole představující nultý řádek tabulky, první prvek obsahuje pole představující první řádek tabulky, ..., lze popsat celou tabulku. Tabulka však není nic jiného než dvourozměrné pole, takže předchozí věta dává návod k jeho tvorbě. Následující kód je ukázkou tvorby dvourozměrného pole T ekvivalentního tabulce T.

```
var T = new Array()
T[0] = new Array()
T[0][0]='a'; T[0][1]='b' ; T[0][2]='c'; T[0][3]='d';
T[1] = new Array()
T[1][0]='e'; T[1][1]='f'; T[1][2]='g'; T[1][3]='h';
```

První index pole T udává řádek a druhý index sloupec tabulky. Příkaz přiřazující jednotlivým prvkům pole hodnotu je totožný zkrácenému zápisu zavedenému pro tabulky. Z toho intuitivně plyne, že pole T a tabulka T popisují totéž.

Zapamatujte si, že dvourozměrné pole se dá chápat jako jednorozměrné pole v němž každý prvek obsahuje zase jednorozměrné pole, stručně řečeno pole polí. Jeho konstrukce je založena, jak jste viděli výše a uvidíte v příkladu níže, právě na tomto faktu.

### Příklad:

Funkce, která vrací dvourozměrné pole (matici) o r řádcích a s sloupcích. Hodnota všech prvků je prázdný řetězec.

*Výpis pole.js*

```
function D2pole(r,s) { /* tvoří pole o r řádcích a s sloupcích */
  var p=new Array()
  for (var i=0; i<r; i++) {           // cyklus definující řádky
    p[i]=new Array()                   // i-tý prvek pole p je opět pole
    for (var j=0; j<s; j++) {         // všem prvkům v i-tém řádku
      p[i][j]=''                     // přiřad ''
    }
  }
  return(p)
}
```

Když si prohlédnete zdrojový kód funkce, zjistíte, že je to pouze přepis myšlenky řečené před tímto příkladem do jazyka JavaScriptu. Pro snadnější pochopení je dobré si místo dvourozměrného pole představit tabulku. S touto funkcí jsem vám vyřešila problém tvorby dvourozměrných

polí. Stačí si ji zkopírovat do skriptu a používat. Jak? No přece do proměnné pouze přiřadit výsledek volání této funkce s patřičnými parametry. Proměnná pak bude pole požadovaných rozměrů. Tedy např.

```
var a=null  
a=D2pole(4, 8)
```

je definice dvojdimenzionálního pole 4 krát 8. Příkaz a[0][0]=4 pak přiřazuje prvku s indexy nula nula hodnotu 4. Sami teď vidíte, že opravdu není problém pomocí jednorozměrných polí vytvořit dvourozměrné. Podobným způsobem by se postupovalo i při tvorbě vícerozměrných polí.

### String (řetězec)

Jedná se o „nejvestavenější“ objekt ze všech vestavěných objektů. Při vytváření řetězcových objektů se nepoužívá slovo new. Ve skutečnosti je každá proměnná, jejíž hodnotou je řetězec, řetězcovým objektem. Má tedy všechny vlastnosti a metody objektu String.

Objekt String má vlastnost length udávající délku řetězce.

Z metod jsou nejzajímavější ty, které pracují s obsahem řetězce. Metoda charAt(index) vrací znak řetězce na pozici index, substring(i,j) celý podřetězec od i-té do j-té pozice. JavaScript indexuje jednotlivé znaky řetězce od nuly (ne od jedničky). Metody indexOf(s) a lastIndexOf(s) vyhledávají podřetězec s v řetězci. Metoda indexOf(s) postupuje od začátku a lastIndexOf(s) od konce řetězce. Při vyhledání podřetězce vrací index jeho prvního znaku v řetězci. V případě neúspěchu obdržíme -1. Poslední metoda, o které chci mluvit, je split(s) rozdělující řetězec do pole prvků na základě předávaného znaku oddělovače. Nemusíte tedy např. číselnou posloupnost načítat číslo po číslu. Stačí načíst vhodný řetězec a ten pak pomocí metody split(s) rozdělit do pole. Následující příklad je důkazem.

### Příklad:

Skript načte po stisknutí tlačítka číselnou posloupnost z textového pole a v dalším textovém poli zobrazí součet posloupnosti. Čísla jsou od sebe navzájem oddělena čárkou a mezerou.

*Výpis split.html(pouze samotný JavaScript)*

```
function Secti(vs) { /* sečte čísla ve vstupním řetězci vs */  
    var a=new Array()  
    var s=0 // součet  
    a=vs.split(", ") // rozdelení řetězce do pole, prvky a jsou řetězce  
  
    for (var i=0; i<a.length; i++) { // pro všechny prvky pole a  
        a[i]=+a[i] // řetězce "konvertuje" na číslo  
        if (a[i]!=='NaN') { // pokud řetězec neobsahoval pouze číslo  
            alert("Zadání není požadovaného tvaru.")  
            return('') // vrátí prázdný řetězec  
        }  
        s+=a[i] // počítání součtu  
    }  
    return(s) // vrátí součet  
}  
  
function Init() { /* zobrazení HTML stránky */  
    var s=''
```

```

s+='<FORM name="f">'
s+='<INPUT type="text" name="vst" size="100">'
s+='<BR> <BR>'
s+='<INPUT type="button" value="Součet" '
s+='onClick="document.f.vystup.value=Secti(document.f.vst.value)">'
s+='<INPUT type="text" name="vystup" size=20>'
s+='</FORM>'
document.write(s)
}

Init()

```

`Init()` zobrazuje HTML stránku s dvěma textovými poli a tlačítkem. Po stisknutí tlačítka je hodnotě textového pole `vystup` přiřazen výsledek `Secti(document.f.vst.value)`. Funkce nejprve rozdělí `vstup` na jednotlivé prvky pole, které by měly obsahovat číselné řetězce. Násobením jedničkou z nich udělá čísla a ty seče. Skript je ukázkou toho, jak lze elegantně vyřešit načítání číselné posloupnosti do pole.

## Math

Objekt `Math` se používá při různých matematických kalkulacích. Tento objekt je zvláštní tím, že netvoříme žádné jeho instance, pouze ho používáme.

Objekt `Math` má vlastnosti zpřístupňují nám standardní matematické konstanty jako je `E`, `PI`, `LN2`, `LN10`. Např. `Math.PI` vrací `PI` atd.

Metodami objektu `Math` jsou běžné trigonometrické a algebraické funkce. Sinus počítá `sin(arg)`, kde parametr je úhel zadáný v radiánech. Inverzní funkci představuje `asin(n)`, kde `n` je číslo náležící do oboru hodnot sinu. Výsledkem je úhel v radiánech. Obdobně pracují i `cos(arg)`, `tan(arg)` a funkce k nim inverzní `acos(n)`, `atan(n)`. Metoda `pow(b, n)` umocní číslo `b` číslem `n`. Jedná se o klasické `b` na `n`-tou. Metoda `sqrt(b)` naopak navrací druhou odmocninu svého argumentu. Objekt `Math` pamatuje i na práci s čísly. Metody `ceil(n)`, `floor(n)`, `round(n)` akceptují při zadávání desetinná čísla, přičemž výstupem je celé číslo. První vrací nejmenší celé číslo, které je větší nebo rovno jejímu argumentu. Druhá vrací největší celé číslo, které je menší nebo rovno jejímu argumentu. Poslední je běžné zaokrouhlování. Např. `ceil(2.4)` je 3, `floor(2.4)` je 2 a `round(2.4)` je 2. Aby toho nebylo málo, `Math` umí generovat pomocí `random()` i náhodná čísla v rozmezí 0 až 1.

## Příklad:

Skript vygeneruje náhodné celé číslo v rozsahu 0 až `n-1` a to zobrazí pomocí alert okénka.

*Výpis random.html (pouze samotný kód JavaScriptu)*

```

var n=10          // největší vygenerované číslo bude n-1
var vysledek=0    // vygenerované číslo

vysledek=Math.floor(n*Math.random())
alert(vysledek)

```

`Math.random()` vygeneruje náhodné číslo z intervalu (0,1). Vynásobením číslem `n` dostaneme číslo z intervalu (0,`n`), které je desetinné. `Math.floor` odřízne desetinnou část a zbude celé číslo v rozsahu od 0 do `n-1`. Pamatujte si, že při generování náhodných celých čísel zpravidla nepoužíváme zaokrouhlování. V tomto případě by vracelo celé číslo v rozsahu od 0 do `n`. Číslo 0 by

vzniklo zaokrouhlením čísel z intervalu (0, 0.5), 1 z intervalu <0.5, 1.5), .... Číslo 0 i n má poloviční interval, než všechna jiná čísla, takže také pravděpodobnost jejich vygenerování je poloviční. Jestliže chcete generovat náhodná celá čísla „spravedlivě“, nikdy výsledek random nezaokrouhlujte!

## Date

Objekt Date zahrnuje práci s datem a časem. Objekt Date JavaScriptu využívá standard z Unixu, používaný pro interní uchovávání data a času jako počet milisekund od 1. ledna 1970. Právě proto neumožňuje pracovat s daty před rokem 1970. Jestliže se o to pokusíte, mohou se vyskytnout neočekávané chyby.

U objektu Date je nejprve nutné si vytvořit jeho konkrétní instanci. To lze udělat hned několika způsoby. Instance s aktuálním časem a datem se získává pomocí new Date(). Příkaz new Date(rok, měsíc, den, hodiny, minuty, sekundy) tvoří instanci představující datum zadané pomocí jeho parametrů. Poslední tři parametry jsou nepovinné. Parametr měsíc má rozsah 0 až 11 a ne 1 až 12 jak byste asi očekávali. Použití tohoto příkazu ozřejmuje instance date1, date2 a date3.

```
date1 = new Date(1990, 0, 29)           // 29. leden 1990 0:00:00  
date2 = new Date(1999, 1, 4, 13)         // 4. únor 1999 13:00:00  
date3 = new Date(1980, 11, 1, 4, 5, 2)    // 1. prosinec 1980 4:05:02
```

Objekt Date nemá ani jednu vlastnost, ale zato velké množství metod vztahující se k datumu a času. Datum se skládá z roku, měsíce a dne a čas zase z hodin, minut a sekund. Rok se řekne anglicky year, měsíc month, hodiny hours, minuty minutes a sekundy seconds. Metoda getYear() vrací číslo reprezentující rok a setYear(rok) naopak mění dosavadní hodnotu roku na rok, getMonth() vrací číslo reprezentující rok a setMonth(měsíc) mění dosavadní hodnotu měsíce na měsíc atd. Den v měsíci lze získávat, resp. měnit pomocí getDate(), resp.setDate(den). Pozor na chyby ústící z toho, že date znamená česky datum a nikoliv den! Například date1.getDate() vrací číslo 29. Další užitečnou metodou je metoda getDay(), která vrací den v týdnu. 0 je ekvivalentem neděle, 1 pondělí atd. Pokud vás zajímá, na který den připadají vaše letošní narozeniny, vytvořte si instanci Date obsahující patřičné datum a použijte na ni metodu getDay(). Poslední metodou, o které se zmíním, je getTime(), která vrací počet milisekund od 1.1.1970. Hodí se například pro srovnání dvou dat, když je úkolem zjistit, které je starší.

### Příklad:

Skript zobrazuje v textovém okně jdoucí digitální hodiny.

```
Výpis clock.html  
<HTML>  
<HEAD>  
  <TITLE> Hodiny </TITLE>  
  
<SCRIPT LANGUAGE="JavaScript">  
<!--  
  
function showtime () /* zobrazuje aktuální čas v textovém poli */  
var now = new Date();           // aktuální datum a čas  
var hours = now.getHours();    // hodiny  
var minutes = now.getMinutes(); // minuty
```

```

var seconds = now.getSeconds()      // sekundy
                           // time Value je řetězec času
var timeValue = "" + ((hours >12) ? hours -12 :hours)
timeValue += ((minutes < 10) ? ":0" : ":") + minutes
timeValue += ((seconds < 10) ? ":0" : ":") + seconds
timeValue += (hours >= 12) ? " P.M." : " A.M."

document.clock.face.value = timeValue;
setTimeout("showtime()",1000);
}

// -->
</SCRIPT>

<NOSCRIPT>
Tento dokument obsahuje kód JavaScriptu.
</NOSCRIPT>

</HEAD>

<BODY onLoad="showtime()">
<form name="clock">
<input type="text" name="face" size=12 value="">
</form>
</BODY>

</HTML>

```

Po natažení dokumentu je spuštěn kód funkce showtime(). V jeho prvním řádku je deklarována instance objektu Date. Následuje deklarace proměnných, které obsahují aktuální hodiny, minuty a sekundy získané z instance objektu. Do proměnné timeValue se ukládá řetězec, jež je vzápětí zobrazen v textovém okně. Při jeho tvorbě je použit ternární operátor k vyřešení problému spojeného s formátem času. U digitálních hodinek je zvykem zobrazovat minuty i sekundy pomocí dvou cifer, proto pokud je minutes, resp. seconds menší než deset skript přičítá do timeValue řetězec ":0" a až pak hodnotu proměnné. Naopak pokud je hodnota minutes, resp. seconds větší rovna deseti (podmínka není splněna) skript přičítá do timeValue pouze řetězec ":" a hodnotu proměnné. Sami si rozmyslete, k čemu slouží ternární operátory spojené s proměnnou hours. Takže teď je vytvořena hodnota timeValue obsahující aktuální čas a zobrazena v textovém okně. Protože cílem příkladu bylo vytvořit jdoucí hodiny, bylo by vhodné v textovém poli zobrazovat aktuální čas vždy po určitém časovém intervalu. Právě o to se stará poslední příkaz funkce, který vždy po tisíci milisekundách tuto funkci spustí znovu.

### Cvičení:

- 1) S použitím metody objektu document write vytvořte tabulkou se čtyřmi řádky a sloupcí, která obsahuje čísla od 0 do 15 (v každém políčku je jedno). Totéž udělejte bez použití JavaScriptu. (*tabulka.html*)
- 2) Vytvořte skript zobrazující confirm okénko s nápisem "Souhlasíš? ". Při stisku OK resp. Storno zobrazí alert s nápisem "Ano! " resp. "Ne! ". (*confirm.html*)

- 3) Vytvořte stránku s výběrovým polem SELECT o položkách slovník, tabulka a faktoriál. Výběr položky a následné opuštění menu způsobí natažení stránky slovnik.html, tabulka.html nebo faktor.html do aktuálního okna. (*select.html* jen IE, *select1.html*)
- 4) Vytvořte stránku s vědomostním nebo psychologickým testem. U každé otázky může uživatel zaškrtnout právě jednu odpověď. Odpovědi jsou hodnoceny body. Po kliknutí na tlačítko zobrazte s využitím příkazu write výsledek testu. (*test.html*)
- 5) Vytvořte stránku s textovým polem a tlačítkem. Po stisknutí se načeť hodnota textového pole a pokud to bude číslo, na stránce se tučným písmem zobrazí všechna prvočísla menší než toto číslo. Pokud žádné prvočíslo nebude existovat zobrazí se nápis informující o tomto faktu. Svůj skript otestujte pro vstupní hodnoty 100, 100000 a srovnajte doby potřebné k odpovědi. (*prvoc.html*)
- 6) Vytvořte stránku, která umožňuje zobrazit textový graf na základě čísel zadaných uživatelem do textových polí. Textových polí je deset a rozsah čísel je 0 až 30 včetně. Graf může vypadat např. jako tabulka, kde každý řádek obsahuje patřičný počet hvězdiček. Při zadávání čísel zobrazujte aktuální součet právě zadaných čísel a jejich průměr. Dále umožněte uživateli vymazat veškerá vstupní data kliknutím na tlačítko. (*textgraf.html*)
- 7) Vytvořte stránku s textovým polem obsahujícím nějaký text, který bude hned při jejím zobrazení označen. (*oznac.html*)
- 8) Vytvořte stránku s animací obrázků od 0 do 9 (najdete je v podadresáři OBRAZKY adresáře cvičení), tj. neustále zobrazujte postupně čísla od 0 do 9. (*animace1.html*)
- 9) Napište skript umožňující načíst číselnou posloupnost a po kliknutí na tlačítko s názvem třídící metody ji setřídit. Z třídících algoritmů postupného hledání minima, bublinového, Insertsortu a Quicksortu nabídněte uživateli nejméně dva. Pokud jste se s třídícími algoritmy doposud nesetkali, vymyslete alespoň dva různé postupy jak třídit číselné posloupnosti. Metodou write vypisujte nejen výslednou setříděnou posloupnost, ale i vhodné mezinásledky s jejichž pomocí je možné pochopit podstatu použitého třídícího algoritmu. (*tralgor.html*)
- 10) Vytvořte skript umožňující načíst čtvercovou matici čísel o maximálně deseti řádcích a po kliknutí na tlačítko
- prohodit její i-tý a j-tý řádek.
  - prohodit její i-tý a j-tý sloupec.
  - vypočítat součet prvků na hlavní a vedlejší diagonále.
  - překlopit její prvky podle hlavní diagonály

Po provedení vybrané akce zobrazte výslednou matici, u c) pomocí alert hodnoty součtu. S výslednou maticí musí jít pracovat stejně tak, jako s původní maticí, tj. uživatel musí mít možnost kliknutím na tlačítko prohodit dva sloupce nebo řádky, vypočítat součty diagonál či překlopit ji podle diagonály. (*matice.html*)

- 11) Napište následující funkce pracující s řetězci a pomocí vhodného skriptu si ověrte jejich funkčnost. (*string.js*)
- Del(s,st) z řetězce s odstraní všechny podřetězce st. Např. Del("NanaNaja", "Na") vrátí "naja", Del("NanaNaja", "a") vrátí "NnNj" atd..
  - Insert(s,st,n) do řetězce s vloží řetězec st od pozice n. Např. Insert("12","a b c ",0) vrátí "a b c 12" atd..
  - rev(s) obrátí řetězec tak, jako by jste ho četli pozpátku. Nepoužívejte žádnou novou proměnnou k ukládání mezivýsledků. Např. rev("123") vrátí "321" atd..
- 12) Vytvořte stránku, která umožňuje hrát opakovaně hru kámen, nůžky, papír člověk proti počítači. (*knp.html*)
- 13) Vytvořte stránku, která umožňuje hrát dvěma hráčům piškvorky. V případě vítězství jednoho hráče vypište informativní zprávu kdo vyhrál. (*pisk.html*)
- 14) Vytvořte skript na procvičování malé násobilky. Na stránce zobrazte 10 náhodně vygenerovaných příkladů. Výsledky bude uživatel vepisovat např. do textových polí za příklady. Předpokládejte, že násobení číslem 0 a 10 nečiní velké potíže a proto ho není nutné tolik procvičovat jako zbytek násobilky. Správnost výsledků bude zkontovalována po stisknutí tlačítka KONTROLA. (*nasob.html*)
- 15) Vytvořte skript na procvičování sčítání a odečítání čísel od 0 do 10. Na stránce zobrazte 10 náhodně vygenerovaných příkladů. Výsledky bude uživatel vepisovat např. do textových polí za příklady. Příklady mohou být tvaru  $5 - ?? = 2$ ,  $5 - 2 = ??$  a  $5 + 4 = ??$ . Nesmí obsahovat příklad  $0 + 0$  a  $0 - 0$ . Správnost výsledků bude zkontovalována po stisknutí tlačítka KONTROLA. (*pocitani.html*)
- 16) Vytvořte HTML stránku rozdělenou na dva framy. V prvním framu zobrazte textové pole a tlačítko. Do textového pole uživatel zadá barvu (jedno zda anglicky nebo hexadecimálně), kterou bude mít po kliknutí na tlačítko pozadí druhého framu. Do druhého framu zobrazte stránku tabulka.html. (*frame.html, menu.html, tabulka.html*)
- 17) Vytvořte stránku obsahující jdoucí digitální hodiny. Jednotlivé znaky hodin zobrazujte jako obrázky (najdete je v podadresáři OBRAZKY adresáře CVICENI). (*anim\_cl.html*)
- 18) Napište funkci, která ve stavovém řádku prohlížeče zobrazí aktuální datum obsahující číslo dne a název měsíce. (*sdate.html*)
- 19) Vytvořte stránku, která po zadání dne a měsíce zobrazí počet dní do zadaného datumu a slovně vypíše název dne v týdnu na který toto datum připadá. (*day.html pouze IE*)
- 20) \* Vytvořte stránku, která umožňuje hrát n hráčům pexeso (obrázky najdete v podadresáři PEXESO adresáře CVICENI). Pokud budete chtít rozšířit tuto stránku ještě o možnost hry člověka proti počítači. (*index.html v adresáři PEXESO*)

# Co se nevlezlo jinam

Tato kapitola je rozdělena na dva celky, které spolu tematicky vůbec nesouvisí. První vás seznámí s některými vestavěnými funkciemi JavaScriptu a druhý ukáže možnosti využití metody `toString()`. Cílem obou částí je rozšířit váš komfort při psaní skriptů. Jedním příkazem budete moci vykonat to, co byste jinak museli popsat svým kódem.

## Vestavěné funkce

Vestavěné funkce jsou, zrovna tak jako vestavěné objekty, přímou součástí JavaScriptu. Jedná se o funkce, které definovali autoři JavaScriptu a na nás je pouze naučit se je vhodně využívat. Pokud byste otevřeli knihu popisující standardizovaný JavaScript našli byste jich šest a to `eval(x)`, `parseInt(string, radix)`, `parseFloat(string)`, `escape(string)`, `unescape(string)`, `isFinite(number)` a `isNaN(number)`. V této statii vám řeknu několik slov o prvních dvouch.

Vestavěná funkce `eval(x)` vrací výsledek matematického výrazu `x`. Například `eval(5+4)` vrací 9. Tuto funkci oceníte zejména pokud se pustíte do tvorby kalkulačky v JavaScriptu. Nebýt jí museli by jste si asi při zadávání pamatovat strukturu matematického výrazu, tj. kde byl zadán jaký operátor a umístění závorek. Na základě těchto informací by jste pak voláním funkcí na sčítání, odečítání, ..., vypočítali výsledek. S funkcí `eval(x)` vám stačí načíst celý matematický výraz a vyhodnotit jej.

Vestavěná funkce `parseInt(string, radix)` vyhodnotí řetězec obsahující celé číslo o základu `radix` a vrátí celé číslo o základu 10. Základ `radix` je nepovinný parametr. Pokud řetězec začíná znakem 0, JavaScript předpokládá, že je v osmičkové soustavě a pokud začíná 0x v šestnáctkové soustavě. Tuto funkci, která umožňuje převod čísla v jakékoliv soustavě do desítkové, by jste si při troše šikovnosti mohli naprogramovat sami.

### Příklad:

Načte `x` a pokud to je matematický výraz zobrazí pomocí `alert` výsledek.

*Výpis eval.html*  
`var x=1*prompt("Zadej x","");
alert(eval(x))`

Jedná se o typickou ukázku funkčnosti `eval(x)`. Načítaný výraz `x` může obsahovat čísla, běžné operátory +, -, \*, /, % (modulo) a závorky. Nebýt této vestavěné funkce, byl by kód realizující zadání rozhodně poněkud obsáhlější.

## Metoda `toString()`

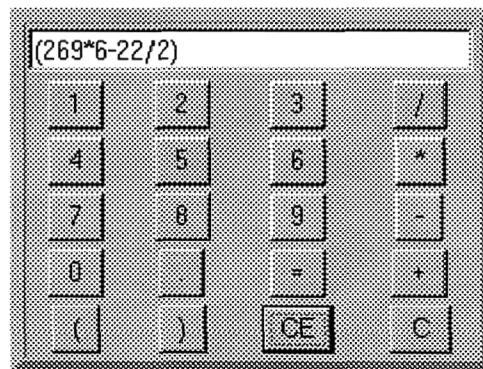
Metoda `toString()` je zajímavá tím, že je to metoda každé proměnné `p`. Proměnná může být samozřejmě i pole či instance objektu `Date`. Příkaz `p.toString()` převede obsah `p` na řetězec. Pokud je `p`

řetězec, vrací přímo hodnotu p, pokud je p pole, vrací stejný výsledek jako p.join() tj. řetězec prvků pole navzájem oddělených čárkou, pokud je p instance objektu Date, vrací řetězec reprezentující datum v p. Jestliže je proměnná p číslo, je možné zadat metodě nepovinný parametr radix. V tom případě bude vrácen řetězec obsahující číslo o zadaném základu radix.

Jedná o první metodu, s kterou jste se setkali v této knize, pracující nad daty různých typů. Její výsledek je tedy závislý nejen na obsahu p, ale i na datovém typu p. Sami si ověřte, že tomu tak skutečně je.

### Cvičení:

- 1) Vytvořte stránku s kalkulačkou. Kontrolujte syntaxi matematického výrazu tvořeného uživatelem a nedovolte mu předat k vyhodnocení nesmyslný výraz. (*kalkul.html*)



- 2) Vytvořte stránku, která umožňuje převádět zadané číslo do soustavy o základu 2, 8, 10, 16. Zadané číslo může být v soustavě o základu 2, 8, 10, 16 a uživatel musí mít možnost vybrat si soustavu do které ho chce převést. V případě zadání nepřípustného vstupu o tom uživatele informujte a dovolte mu opravit právě zamítnutou hodnotu. (*prevod.html*)

## Zdroje JavaScriptu

V kurzu, který jste právě absolvovali, jste se aktivně seznámili s mnoha možnostmi JavaScriptu. Cvičení vám dala prostor k vytvoření několika desítek skriptů. Nenásilně jste se tak naučili s pomocí JavaScriptu řešit jednoduché i složitější zadané problémy. Při troše fantazie dokážete s využitím vhodné kombinace získaných znalostí navrhnout a realizovat svoje vlastní dynamické stránky. Jelikož vám tato publikace nabídla pouze základy JavaScriptu a JavaScript je stejně jako všechno, co je spojeno s webem, stále se vyvíjející jazyk, je dobré si rozšířit rozhled o další informace. Dle vlastního uvážení si můžete nakoupit knihy, či si vyhledat materiály na webu. Pokud preferujete českou literaturu, pak zejména u knih musíte počítat s relativní zastaralostí jejich obsahu a u většiny webových dokumentů zase s obsahovou zkresleností vyplývající z nepřesného překladu anglického originálu. Proto, jestliže umíte anglicky, čtěte materiály o JavaScriptu i, nebo pouze, v originále.

Následující řádky obsahují adresy dokumentů, které by vás mohly zajímat. Jsou rozdeleny do dvou částí podle jazyka v němž je dokument prezentován. Veškeré doporučované stránky byly aktivní a spolehlivé 10. 11. 2000. Protože se obsah webu neustále mění, nemohu vám zaručit jejich existenci či tématický obsah v době, kdy listujete touto publikací.

### Česky a slovensky psané HTML stránky

<http://cicinovo.misto.cz/javascript/index.html>

Jedná se o bakalářskou práci Petra Kočičky vytvořenou v roce 1996. Popisuje JavaScript ve verzi, která tehdy existovala. Základy jsou ale stejné a jistě zde najdete mnoho užitečných rad. S laskavým svolením autora ji můžete najít také na přiloženém médiu v adresáři KOCICKA.

<http://www.javascript.sk>

Na těchto stránkách se nachází jeden z nejrozsáhlejších archívů JavaScriptu. Můžete se tak inspirovat již existujícími skripty k tvorbě svých vlastních. Kromě toho zde můžete najít manuál, který je určen úplným začátečníkům, co se týče JavaScriptu.

<http://java.tatousek.cz/>

Mimo archív skriptů, zde najdete adresu e-mailové konference o tvorbě WWW stránek a diskusní fórum se stejnou tematikou.

<http://www.angelfire.com/mt/komarek/>

Tyto stránky obsahují pář užitečných informací o tvorbě WWW stránek. Dávají k dispozici několik JavaScriptů, které můžete volně využít pro své stránky. Najdete zde odkazy i na jiné archívy (JavaScript, CGI skripty, obrázky a animace, software), odkazy na dokumentaci o HTML, JavaScriptu atd.

[http://members.nbcu.com/\\_XMCM/v\\_p/index.htm](http://members.nbcu.com/_XMCM/v_p/index.htm)  
<http://www.eecs.umich.edu/~bartlett/javascript.html>

Jedná se o dva manuály JavaScriptu, z nichž ani jeden nejde do patřičné hloubky. Chcete-li si však osvojené informace konfrontovat i s jinými zdroji, tyto stránky vám k tomu dávají příležitost.

## Anglicky psané HTML stránky

<http://developer.netscape.com/docs/manuals/>

Firma Netscape vyráběla první prohlížeč podporující JavaScript. Pokud vás zajímají aktualizace a dodatky JavaScriptu, stránky Netscapu jsou ty, které hledáte. Na uvedené adrese najdete odkazy na velké množství manuálů. Jedním z nich je manuál JavaScriptu.

<http://webteacher.com/javascript>

Zde je umístěn tutoriál JavaScriptu.

<http://www.javascriptsource.com>

Pokud si myslíte, že jste stále nenašli potřebné informace o JavaScriptu, zkuste se podívat na tuto stránku. Obsahuje zejména informace či odkazy na dané téma.

<http://www.infohiway.com/javascript/indexf.htm>

Na těchto stránkách je kromě jiného umístěn rozsáhlý archív JavaScriptů.

<http://www.javascripts.com>

I tato adresa vám nabízí velké množství skriptů.

Předchozí řádky představují jen ty, podle mého názoru, nejdůležitější dokumenty pojednávající o JavaScriptu. Další stránky s touto tématikou lze najít s pomocí vyhledávačů (Seznam, AltaVista atd.). Zadané slovo, na jehož základě je obsah webu prohledáván, může být například JavaScript,

ECMAScript či ECMA-262. ECMAScript je totiž standardizovaný JavaScript a ECMA-262 samotný standard. ECMAScript je sice o něco málo „chudší“ než JavaScript, ale zato skripty napsané pouze pomocí ECMAScriptu fungují na všech prohlížečích podporujících JavaScript, což se o běžných Scriptech říci nedá. Vlastně celá tato publikace, na jejímž konci se nacházíte, popisovala část průniku mezi ECMAScriptem a JavaScriptem.

Ještě bych vás přece jen chtěla upozornit na jednu webovou adresu a to:

<http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM>

Najdete zde dokumenty popisující ECMAskript.

A to už je ode mne opravdu vše.

# Rejstřík

anchors.....	49	vnořený.....	39
array.....	53	objektově orientované programování .....	33
break .....	23	objekty JavaScriptu.....	45
continue.....	23	hierarchie .....	46
cyklus		HTML .....	48
for.....	22	prohlížeče.....	45
for...in.....	41	vestavěné.....	53
while .....	21	odladování.....	9
date .....	58	operátor.....	16
datový typ.....	15	priorita .....	16
explicitní .....	18	podmínka .....	20
implicitní .....	18	pole .....	38,53
document.....	46	asociativní .....	38,54
forms.....	49	jednorozměrné.....	54
frames .....	47	vícerozměrné.....	54
funkce .....	24	procedura .....	24
rekurzivní .....	24	prohlížeč webových stránek .....	7
vestavěné .....	63	proměnná .....	14
history .....	48	globální .....	24
if...else .....	21	lokální .....	24
images.....	49	příkaz .....	15
JavaScript.....	6	řídící struktura .....	20
syntaxe .....	14	SCRIPT.....	12
komentáře.....	9	skriptovací jazyk .....	9
konverze.....	18	stránka .....	
links .....	49	dynamická .....	6
location .....	48	statická .....	6
math.....	57	string .....	56
NAME.....	51	ternární operátor .....	21
new .....	37	this .....	34,52
objekt .....	33	událost .....	29
deklarace .....	34	výraz .....	16
instance .....	34,37	window .....	46
schopnost (metoda) .....	33	with .....	42
vlastnost .....	33	základní kostra dokumentu.....	12

## Příloha A

### Rezervovaná slova JavaScriptu

Rezervovaná slova JavaScriptu nelze použít jako název proměnné či funkce. Pokud toto pravidlo porušíte váš skript bude pracovat neočekávaným způsobem a nebo nebude pracovat vůbec.

Rezervovaná slova se dělí na klíčová slova a slova o kterých se předpokládá, že by se klíčovými mohla někdy v budoucnosti stát.

Klíčová slova:

break	for	new	var
continue	function	return	void
delete	if	this	while
else	in	typeof	with

Ostatní rezervovaná slova:

abstract	do	import	short
boolean	double	instanceof	static
byte	enum	int	super
case	export	interface	switch
catch	extends	long	synchronized
char	final	native	throw
class	finally	package	throws
const	float	private	transient
debugger	goto	protected	try
default	implements	public	volatile

## Příloha B

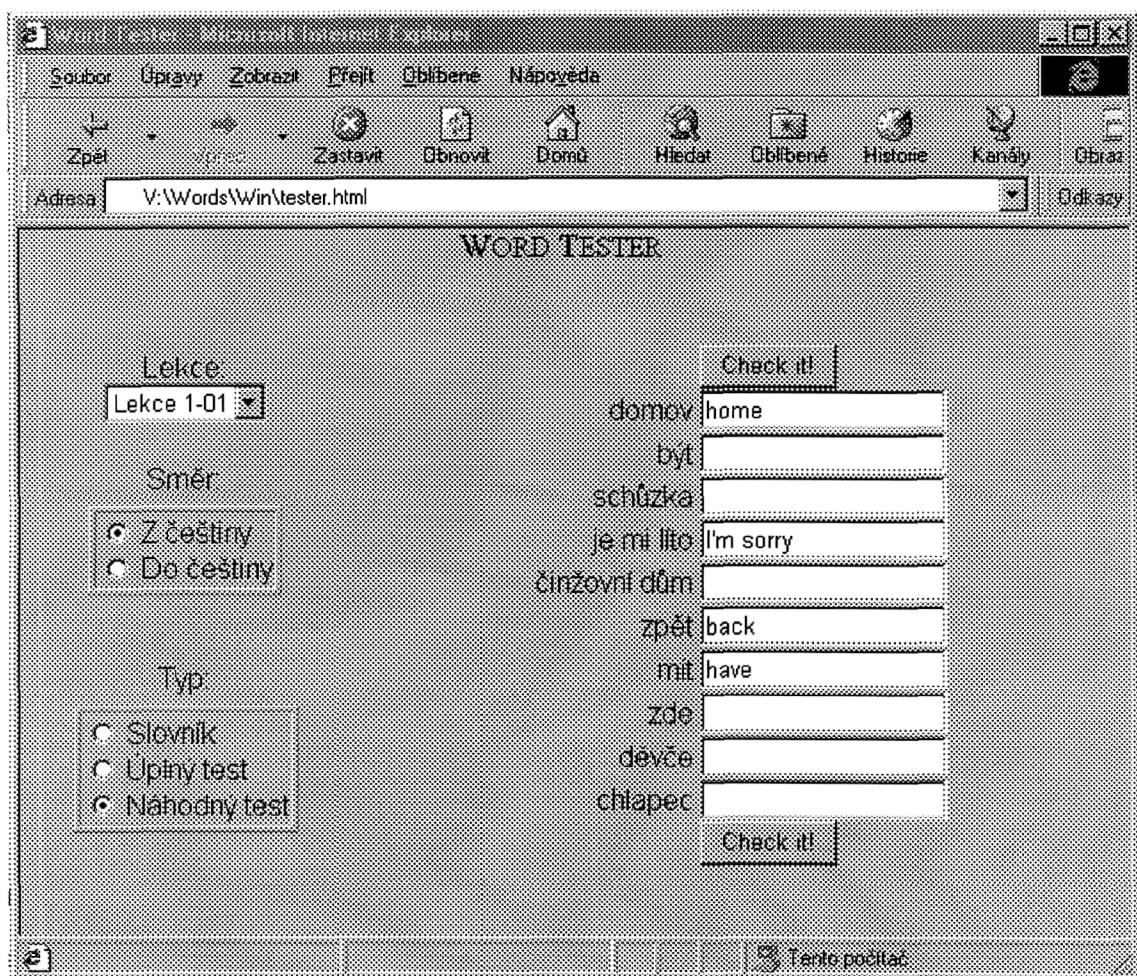
### Ukázka použití JavaScriptu v praxi

V adresáři WORDS najdete složitější aplikaci vytvořenou pouze s pomocí JavaScriptu. Jedná se o slovníček a stránku umožňující testovat znalost slovíček obsažených v učebnici Angličtina pro jazykové školy I. V adresáři WIN je verze pro Windows a v UNIX verze pod Unix. Obě se od sebe liší pouze použitým kódováním.

#### Pohled pro uživatele

Otevřením souboru finder.html se vám zobrazí stránka, která pracuje jako slovníček. Do textového pole napište slovíčko, jež chcete přeložit, zaškrtněte jazyk do něhož se bude překládat a stiskněte tlačítko. Při nalezení žádaného překladu je překlad včetně lekce výskytu slovíčka vypsán na obrazovku. Pokud není slovíčko obsaženo v tomto mini slovníčku na stránce se objeví nápis informující vás o tomto faktu. Tento skript funguje pod oběma prohlížeči.

Po otevření souboru tester.html v prohlížeči budete moci testovat svoji znalost slovíček. Jeho vzhled vidíte na obrázku na následující straně. Výběrové pole nadepsané lekce slouží k výběru právě jedné lekce jejíž slovíčka si chcete procvičit a nebo položky Mistakes či Cookies. Tyto položky vám umožňují procvičit si znova slovíčka v nichž jste udělali chybu při dnešním či minulém sezení. Zaškrnutím směru si volíte směr překladu, tzn. zda chcete překládat z češtiny nebo do češtiny. Můžete si také zvolit zda máte zájem o výpis slovníčku nebo o úplný či náhodný test. Zobrazená stránka se aktualizuje podle vašich přání. U testů máte vždy možnost své překlady vepsat do příslušných textových polí a správnost si ověřit stisknutím tlačítka. Bohužel při volbě směru z češtiny právě toto u Netscape Navigatoru nefunguje seč jinak skript v tomto prohlížeči pracuje zcela správně. V Exploreru skript funguje bez jakýchkoliv problémů.



## Pohled pro programátora

Pokusím se vám alespoň mírně nastínit, jak je programově řešena stránka na testování slovíček.

Stránka přímo reaguje na volby lekce, směru a typu. Na základě těchto voleb vygeneruje obsah své pravé části a zobrazí ho. Z toho vám musí být jasné, že ve skutečnosti je stránka rozdělena na frames. Při testování slovíček se do speciálního pole ukládají slovíčka v nichž byla dnes udělana chyba. Po opuštění stránky je obsah tohoto pole uložen do cookies. Jedná se o soubor uložený na počítači uživatele určený k účelu ukládání a následného využívání dat získaných z HTML stránek. Je dosti nespolehlivý a je na něj kladenou několik omezujících podmínek. Rozhodně se nedá předpokládat, že tato technika bude někdy zařazena do standardizovaného JavaScriptu a proto jsem se jí nevěnovala ani v mé knize. Pokud vás i přesto zaujala, nastudujte si ji sami.

Programátor musí ale nejprve vyřešit problém jak reprezentovat jednotlivá slovíčka. Slovíčka jsou totiž základním kamenem tohoto skriptu. Po zralé úvaze jsem se rozhodla slovíčka reprezentovat pomocí dvourozměrného asociativního pole. Jednorozměrné asociativní pole nestačí, protože jedno anglické slovo může mít hned několik překladů a naopak a já si ještě navíc přála vědět u každého slovíčka lekci původu. Další problém, s kterým jsem se v této souvislosti setkala,

spočíval v tom, že slovíčka mi byla dodána v textovém souboru a já je musela teprve nějak dostat do kýženého pole. Poradila jsem si tak, že jsem každou lekci slovíček uložila do souboru s příponou js. Slovíčka jsem navzájem oddělila speciálním znakem a zrovna tak jejich český a anglický význam. V souboru slovíček jsem pak zavedla proměnnou, do níž jsem si všechna slovíčka uložila jako jeden dlouhý řetězec slovíček. Až tento řetězec jsem postupně proměnila v dvouzměrné pole.

Z předchozích dvou odstavečků vám je asi jasné, že k tvorbě tohoto skriptu nebylo zapotřebí pouze znalost JavaScriptu, ale i určité fantazie. Cílem tohoto skriptu nebylo pouze vás přesvědčit, že i programátor musí být člověk s velkou fantazií, ale ukázat vám možnost využití JavaScriptu, kde by jste ho možná ani nehledali a v neposlední řadě vás inspirovat k tvorbě vlastních skriptů.

## Příloha C

### Srovnání JavaScriptu a Pascalu

Většina z vás umí asi programovat v Pascalu a právě proto se vám pokusím mírně usnadnit učení JavaScriptu tím, že ho srovnám v následující tabulce s Pascalem. Za tabulkou najdete ještě srovnání JavaScriptu a Pascalu s ohledem na příkazy obsažené v těchto jazycích.

JavaScript	Pascal
skriptovací jazyk	programovací jazyk
spouští se přímo zdrojový kód	spouští se zkompilovaný zdrojový kód
program nemá předepsanou strukturu, proměnnou lze deklarovat až před místem použití, funkci až před místem volání atd.	za hlavičkou programu následuje definice a deklarace návěští, typů, proměnných, funkcí a procedur a až po nich samotné tělo programu
podporuje událostmi řízené programování	nepodporuje událostmi řízené programování
má vestavěné objekty	nemá vestavěné objekty
lze vytvářet své vlastní objekty	lze vytvářet své vlastní objekty
rozlišuje malá a velká písmena	nerozlišuje malá a velká písmena
deklarace proměnných není povinná	deklarace proměnných je povinná
implicitní datové typy	explicitní datové typy
dynamická pole	statická pole
nelze přímo deklarovat vícerozměrné pole - musíte si je vytvořit sami pomocí jednorozměrných polí	lze přímo deklarovat vícerozměrné pole
neumí pracovat s textovými soubory	umí pracovat s textovými soubory

JavaScript obsahuje jako Pascal řídící struktury if, cyklus while a cyklus for. Příkaz if a cyklus while má v JavaScriptu zcela stejnou sémantiku jako v Pascalu. Cyklus for se sémanticky mírně liší, ale jeho početní síla je v obou jazycích stejná. JavaScript neobsahuje cyklus repeat a podmíněný příkaz case.

V JavaScriptu se pod pojmem funkce skrývají pojmy funkce a procedura v Pascalu.

Třídící algoritmy a velké množství programů v Pascalu, které nepoužívají standardní programové jednotky, lze bez velkých problémů přepsat do JavaScriptu.

## **Použitá literatura a odkazy**

1. M. C. Reynolds, A. Woolridge, JavaScript – profesionální řešení, UNIS Publishing, 1997
2. L. Purcell, M. J. Mara, JavaScript – tvorba dokonalých WWW stránek, podrobný průvodce začínajícího uživatele, Grada Publishing, 1998
3. <http://developer.netscape.com/tech/javascript/index.html>
4. <http://developer.netscape.com/docs/javascript/e262-pdf.pdf>
5. <http://cicinovo.misto.cz/javascript/index.html>
6. <http://www.javascript.sk>
7. <http://www.kosek.cz/clanky/dhtml/skripty.html>
8. Zápisky z předmětu Služby počítačových sítí (P005) přednášeném na FI MU